Technische Universität Braunschweig

From Grown to Structured - Reduzierung unnötiger Variabilität von Technologiearchitekturen in gewachsenen IT-Landschaften

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von
Kenny Wehling
geboren am 10.09.1984
in Berlin

Eingereicht am:	07.01.2019
Disputation am:	12.04.2019
1. Referentin:	Prof. Dr.-Ing. Ina Schaefer
2. Referent:	Prof. Dr.-Ing. Bernhard Rumpe

2019

Erklärung

Die Ergebnisse, Meinungen und Schlüsse dieser Arbeit sind nicht notwendigerweise die der Volkswagen AG.

Declaration

The results, opinions and conclusions expressed in this thesis are not necessarily those of Volkswagen AG.

Danksagung

Die vorliegende Doktorarbeit entstand während meiner Tätigkeit als Doktorand in der Konzern-IT der Volkswagen AG. Die wesentlichen Inhalte wurden dabei im Rahmen eines gemeinsamen Forschungsprojektes mit der TU Braunschweig erarbeitet. An dieser Stelle möchte ich all denjenigen Personen danken, die mich bei der Entwicklung der in dieser Arbeit präsentierten Inhalte unterstützt haben.

Mein erster Dank gilt meiner Betreuerin Frau Prof. Dr.-Ing. Ina Schaefer, die mich in allen Phasen meiner Arbeit begleitet und stets konstruktiv unterstützt hat. Darüber hinaus möchte ich auch Herrn Dr.-Ing. David Wille danken, mit dem ich in unserem gemeinsamen Forschungsprojekt eng zusammenarbeiten durfte und der mich mit wertvollen Diskussionen, hilfreichem Feedback und seiner kompetenten Mitarbeit an den gemeinsam veröffentlichten Konferenz- und Workshopartikeln intensiv unterstützt hat. Mein Dank geht zudem an Herrn Dr.-Ing. Christoph Seidl für seine umfangreiche Unterstützung bei inhaltlichen Fragestellungen und bei der Verfassung von gemeinsam veröffentlichten Konferenz- und Workshopartikeln. Außerdem danke ich Herrn Prof. Dr.-Ing. Bernhard Rumpe für die Erstellung des Zweitgutachtens sowie Herrn Prof. Dr.-Ing. Lars Wolf für die Übernahme des Vorsitzes der Prüfungskommission.

Ebenfalls möchte ich mich bei meinen Kollegen der Volkswagen AG für ihren Rückhalt und ihre umfangreiche Unterstützung bedanken. Insbesondere gilt mein Dank Herrn Matthias Wandert, der alle organisatorischen Rahmenbedingung geschaffen hat, damit ich meine wissenschaftlichen Tätigkeiten im Unternehmen erfolgreich durchführen konnte. Darüber hinaus geht auch ein besonderer Dank an all diejenigen Kollegen, die mich mit ihrer fachlichen Kompetenz in zahlreichen Diskussionen, Fallstudien und Experteninterviews begleitet haben. Stellvertretend möchte ich hier gerne Herrn Arne Stahmer nennen, der mich häufig bei der Erarbeitung von Konzepten mit seinem Fach- und Methodenwissen unterstützt hat.

Abschließend geht auch noch ein besonderer Dank an meine Freundin Sophie Gehr, die mir in meinem privaten Umfeld stets den Rückhalt gegeben hat, damit ich den langen Weg der Promotion erfolgreich meistern konnte.

Zusammenfassung

Die IT-Landschaft in einem Unternehmen entwickelt sich typischerweise über viele Jahre und Jahrzehnte. Um die wachsenden Anforderungen und den steigenden Bedarf an Softwarelösungen zur Unterstützung unterschiedlichster Geschäftsfunktionen (z.B. Logistik, Produktion und Kundenmanagement) zu realisieren, werden so über die Zeit immer mehr Anwendungssysteme geschaffen und in die bestehende Landschaft integriert. Allerdings ist diese Entwicklung häufig unkoordiniert, sodass neue Anforderungen und neue Technologien oftmals mit zusätzlichen System-Lösungen umgesetzt werden, was zu einem kontinuierlichen Wachstum der IT-Landschaft führt.

In der Konsequenz können solche gewachsenen IT-Landschaften aus hunderten oder tausenden von Softwaresystemen bestehen, die ein breites Spektrum unterschiedlichster Technologien verwenden. Obwohl solche Systeme sehr verschieden sein können, setzen viele von ihnen gleiche Kerntechnologien ein (z.B. Java, .Net oder SAP). Allerdings unterscheiden sie sich häufig in anderen eingesetzten technologischen Komponenten (z.B. verschiedene Betriebssysteme, Datenbanksysteme oder Datenintegrationstechnologien). Diese technologischen Varianten sind aus Architektursicht nicht immer erforderlich und verursachen unnötige Variabilität in den Technologiearchitekturen der betrachteten Systeme, was zu einer höheren Komplexität, einer geringeren Anpassungsfähigkeit sowie zu steigenden Kosten und höherem Aufwand für die Wartung und Weiterentwicklung der gesamten IT-Landschaft führt.

Um diesen Herausforderungen zu begegnen, ist es erforderlich, dass die Variabilität von technologisch verwandten Softwaresystemen reduziert wird. Hierzu müssen Experten die Technologiearchitekturen aller in Frage kommender Systeme einzeln analysieren, die beinhaltete Variabilität identifizieren sowie geeignete Restrukturierungspotentiale ableiten und bewerten. Da diese komplexen Aufgaben bisher manuell von Experten durchgeführt werden müssen, sind sie für gewachsene IT-Landschaften mit hunderten von IT-Systemen kaum machbar. Zur Lösung dieses Problems werden in dieser Dissertation drei wissenschaftliche Beiträge vorgestellt: (1) Ein Mining-Verfahren zur Bestimmung von Variabilität in Technologiearchitekturen, welches eine beliebige Anzahl an verwandten IT-Systemen analysiert und alle Variabilitätsbeziehungen zwischen ihnen bestimmt. (2) Ein regelbasierter Ansatz zur Ableitung von Restrukturierungsempfehlungen, welcher unnötige Variabilität in den betrachteten Technologiearchitekturen identifiziert und geeignete Maßnahmen zur Reduzierung dieser Variabilität vorschlägt. (3) Ein Ansatz zur Simulation und Bewertung von abgeleiteten Restrukturierungsempfehlungen, welcher Experten bei der Entscheidungsfindung zur konkreten Restrukturierung von betrachteten Technologiearchitekturen unterstützt. Alle Beiträge wurden mit Experteninterviews und Fallstudien evaluiert. Für diese Evaluation standen uns verschiedene Experten sowie Daten von realen Technologiearchitekturen eines Industriepartners zur Verfügung.

Abstract

A company's IT landscape typically evolves over years and even decades. By satisfying the growing demand for software solutions, the number of software systems increases with a company's requirements to support various business functions (e.g., logistics, production, and customer management). As such an evolution is normally uncoordinated, the realization of new requirements or the implementation of new technologies often results in additional software systems. This leads to a continous growth of an enterprise IT landscape.

As a consequence, grown IT landscapes can consist of hundreds or thousands of different software systems with a large variety of technologies. Although such software systems can be very different, a lot of them typically utilize the same core technology (e.g., Java, .Net, or SAP). However, they often differ in additional technology components that are also required to run a software system (e.g., different operating systems, database systems or data integration technologies). From an architectural point of view, such technical variants are not always necessary and might lead to unnecessary variability in the technology architectures of regarded software systems. This results in increasing costs, a reduced adaptability and higher effort for maintaining and evolving existing software solutions and the entire IT landscape.

To cope with these challenges, the variability of related software systems has to be reduced. For this purpose, domain experts have to analyze the technology architectures of potential software systems. More precisely, they have to identifiy the inherent variability of such software systems and to assess suitable restructuring potentials. As this is a manual and complex task, it is not feasible for a large number of software systems. Thus, experts continuously face the tedious challenge of making reasonable restructuring decisions for large-scale IT landscapes. To solve the described problems, this doctoral thesis comprises three different scientific contributions: (1) An automated mining approach for determining variability in technology architectures, which is capable of analyzing a large number of software systems and determining the inherent variability relations. (2) A rule-based approach for deriving restructuring recommendations, which identifies unnecessary variability in the considered technology architectures and suggests appropriate restructuring measures to reduce this variability. (3) An approach for evaluating and simulating derived restructuring recommendations, which supports experts in taking reasonable decisions for restructuring of analyzed technology architectures. All contributions were evaluated by means of expert interviews and several case studies. For this purpose, we had access to various experts as well as real-word data from our industry partner.

Liste der Veröffentlichungen

Diese Doktorarbeit basiert auf den folgenden Veröffentlichungen, absteigend sortiert nach ihrer Relevanz:

K. Wehling, D. Wille, C. Seidl, I. Schaefer: *Reducing Variability of Technically Related Software Systems in Large-Scale IT Landscapes*, Intl. Conference on Computer Science and Software Engineering (CASCON), **Best Paper Award**, ACM, 2018

K. Wehling, D. Wille, C. Seidl, I. Schaefer: *Automated Recommendations for Reducing Unnecessary Variability of Technology Architectures*, Proc. of the Intl. Workshop on Feature-Oriented Software Development (FOSD), ACM, 2017

D. Wille, **K. Wehling**, C. Seidl, M. Pluchator, I. Schaefer: *Variability Mining of Technical Architectures*, Proc. of the Intl. Systems and Software Product Line Conference (SPLC), **HITACHI Young Best Paper Award – Research Track**, ACM, 2017

K. Wehling, D. Wille, C. Seidl, I. Schaefer: *Decision Support for Reducing Unnecessary IT Complexity of Application Architectures*, Proc. of the Intl. Workshop on decision Making in Software ARCHitecture (MARCH), IEEE, 2017

K. Wehling, I. Schaefer: *Towards an Expert System for Identifying and Reducing Unnecessary Complexity of IT Architectures*, In: M. Eibl, M. Gaedke (Hrsg.), INFORMATIK 2017, Gesellschaft für Informatik, Bonn, 2017

K. Wehling, D. Wille, M. Pluchator, I. Schaefer: *Towards Reducing the Complexity of Enterprise Architectures by Identifying Standard Variants Using Variability Mining*, 1. Automobil Symposium Wildau: Tagungsband, 2016

Inhaltsverzeichnis

Verzeichnis der Tabellen

Verzeichnis der Abbildungen

Verzeichnis der Abkürzungen

ACM Adaptive Case Management.

ADD Architectural Design Decision.

AG Alternativgruppe.

AHP Analytical Hierarchy Process.

AM Anpassungsmaßnahme.

ATAM Architecture Tradeoff Analysis Method.

BDD Behavior Driven Development.

BPMN Business Process Model and Notation.

CEADA Collaborative Evaluation of Enterprise Architecture Design Alternatives.

CVL Common Variability Language.

DBS Datenbanksystem.

DSGVO Datenschutz-Grundverordnung.

DSL Domain-Specific Language.

EA Enterprise Architecture.

EAM Enterprise Architecture Management.

EAMAM Enterprise Architecture Modifiability Analysis Method.

EBNF Extended Backus-Naur Form.

ECA Event Condition Action.

EMF Eclipse Modeling Framework.

EPK Ereignisgesteuerte Prozesskette.

FCA Formal Concept Analysis.

FF Forschungsfragen.

GEF Graphical Editing Framework.

GPL General-Purpose Language.

GR Geschäftsregeln.

HDBS Hierarchisches Datenbanksystem.

HTML Hypertext Markup Language.

IDE Integrated Development Environment.

JRE Java Runtime Environment.

JUNG Java Universal Network/Graph Framework.

MM Migrationsmaßnahme.

MOF Meta-Object Facility.

MoVaC Model Variants Comparison.

NSGA-II Non-Dominated Sorting Genetic Algorithm.

OVM Orthogonal Variability Model.

OWL Web Ontology Language.

PRM Probabilistic Relational Model.

RCP Rich Client Platform.

RDBS Relationales Datenbanksystem.

RE Restrukturierungsempfehlung.

RG Relationsgruppe.

RP Restrukturierungspotential.

SAAM Software Architecture Analysis Method.

SPL Software Product Line.

SPLE Software Product Line Engineering.

TA Technologiearchitektur.

TAP Technologiearchitekturplan.

TCO Total Cost of Ownership.

TOGAF The Open Group Architecture Framework.

VD Variabilitätsdifferenz.

VKG Variantenspezifischer Komponentengraph.

VSM Viable System Model.

XAML eXtensible Application Markup Language.

1 Einführung

Die IT-Landschaft in einem Unternehmen wächst typischerweise über viele Jahre und Jahrzehnte [Han10, DKK+12]. Im Laufe dieser Zeit entsteht eine Vielfalt von verschiedenen Softwaresystemen mit dem Ziel, die sich stetig verändernden Unternehmensanforderungen für unterschiedlichste Geschäftsfunktionen (z.B. Kundenmanagement, Logistik und Produktion) mit Hilfe von individuellen und standardisierten Softwarelösungen umzusetzen [Dur08]. So kann sich eine IT-Landschaft schnell zu einer großen und umfangreichen Infrastruktur entwickeln, welche hunderte oder gar tausende von Softwaresystemen beinhaltet [Sch09, Ben14]. Doch solch eine Entwicklung geschieht in Unternehmen oft unkoordiniert [Der09, ASML12], sodass neue Anforderungen oftmals auch zu neuen Softwaresystemen führen. Die Gründe hierfür sind beispielsweise die fehlende Übersicht über vorhandene IT-Lösungen [ASML12] und die Tatsache, dass eine gewachsene IT-Landschaft typischerweise einem Flickenteppich von Hard- und Softwarekomponenten gleicht, weshalb IT-Architekten und Entwickler davor zurückschrecken, Veränderungen an diesem komplexen Gesamtsystem vorzunehmen. Hierdurch entsteht allerdings ein Teufelskreislauf, der die Komplexität einer gewachsenen IT-Landschaft immer weiter steigen lässt [Han10, BKAV17]. Durch zusätzliche Umstrukturierungen, Integration weiterer Softwaresysteme und IT-Innovationen wird dieser Trend noch weiter verstärkt [Dur08, Han10, ASML12]. So entstehen große, gewachsene IT-Landschaften, die für Unternehmen kaum noch beherrschbar sind [Dur08]. Daher bewerten knapp zwei Drittel aller Unternehmen die Komplexität ihrer Anwendungslandschaft als zu hoch und führen dies unter anderem auf die fehlende Standardisierung beziehungsweise Konsolidierung ihrer IT-Systeme sowie deren Hard- und Softwarekomponenten zurück [Mat15]. Aus diesem Kontext heraus hat sich in der Praxis der Begriff *Spaghetti-Architektur* entwickelt, welcher zur Verbildlichung der komplexen und unentwirrbaren Zusammenhänge einer gewachsenen IT-Landschaft verwendet wird [HHH09]. Auch in der Wissenschaft stellt die steigende Komplexität von IT-Landschaften eine der größten Herausforderungen im Forschungsfeld der *Unternehmensarchitektur* dar [LGv+16].

1.1 Motivation und Problemstellung

Eine Unternehmensarchitektur, auch als *Enterprise Architecture (EA)* bezeichnet, stellt die fundamentale Konzeption aller Elemente und Beziehungen einer Organisation dar [Sch16]. Dabei berücksichtigt diese sowohl die businessorientierte Geschäftsarchitektur als auch die technologieorientierte IT-Architektur. Das Fundament dieser IT-Architektur bildet die sogenannte *Technologiearchitektur (TA)*, welche verschiedene Hard- und Softwarekomponenten (z.B. Betriebssysteme,

Datenbanken und Applikationsserver) beinhaltet, die zum Betrieb von Softwaresystemen (z.B. einem Kundenmanagementsystem) benötigt werden [The11].

Die steigende Komplexität von IT-Landschaften wirkt sich insbesondere auf die Technologiearchitektur von Unternehmen aus. So stellt die Diversität von Technologien einen der wesentlichen Treiber von IT-Komplexität dar [BAHA16]. Diese führt zu einer Vielfalt eingesetzter Technologien, welche immer wieder durch neue Entwicklungen erweitert werden [Tie16]. Doch nur selten findet auch im gleichen Maße eine Entfernung von veralteten und nicht mehr genutzten Technologien aus einem Unternehmen statt [Han11]. So entstehen Redundanzen in den technischen Ausstattungen von Softwaresystemen, welche oft Komponenten mit ähnlichen oder gar identischen Funktionen implementieren [Tie16]. Dies kann beispielsweise häufig bei technisch verwandten IT-Systemen beobachtet werden, welche sich durch die Verwendung der selben Kerntechnologie (z.B. Java, .Net, SAP, etc.) auszeichnen. Hier lässt sich oft feststellen, dass solche Systeme durch unterschiedliche Komponenten realisiert worden sind, obwohl sie im Kern technisch gleich funktionieren. Beispielsweise können sich zwei Softwaresysteme, welche beide *Java* mit dem gleichen Web-Applikationsserver `Tomcat` verwenden, in den eingesetzten Betriebssystem-, Datenbank- oder Authentifizierungskomponenten unterscheiden (z.B. `Oracle DB` versus `IBM DB2`). In der Konsequenz führen solche Varianten zu erhöhter *Variabilität* in den Technologiearchitekturen der betrachteten Softwaresysteme. Dabei wird unter Variabilität im Allgemeinen verstanden, dass ein Softwaresystem an einen spezifischen Kontext angepasst werden kann [GAWM11], beispielsweise auch durch die Verwendung unterschiedlicher Hard- und Softwarekomponenten auf Ebene der Technologiearchitektur. Die sich daraus ergebenen Folgen haben allerdings Auswirkungen auf die gesamte IT-Architektur eines Unternehmens. So steigt beispielsweise der Aufwand für Wartung und Weiterentwicklung von Softwaresystemen. Zudem erhöhen sich die Kosten für die Infrastruktur aufgrund komplexer Integrationsarchitekturen und es verringert sich die Fähigkeit des Unternehmens, die IT-Architektur schnell an veränderte Rahmenbedingungen anpassen zu können (z.B. an die neue EU-Datenschutz-Grundverordnung (DSGVO)[1]) [HHH09, ASML12].

Um diesen negativen Konsequenzen zu begegnen, muss die Variabilität innerhalb einer Technologiearchitektur durch geeignete Umstrukturierungen reduziert werden. Zu diesem Zweck ist es erforderlich, dass Domänenexperten eine gründliche Analyse der Technologiearchitektur eines betrachteten Unternehmens durchführen. Ziel ist es dabei solche technologischen Komponenten zu identifizieren, die für die Umsetzung der Geschäftsanforderungen und -ziele nicht benötigt werden und somit zu *unnötiger Variabilität* führen. Zudem müssen für die so identifizierten unnötigen Varianten auch geeignete Restrukturierungspotentiale bestimmt und bewertet werden, damit Experten in der Lage sind, begründete und nachvollziehbare Entscheidungen zugunsten konkreter Restrukturierungsmaßnahmen treffen zu können.

[1] https://www.bmwi.de/Redaktion/DE/Artikel/Digitale-Welt/europaeische-datenschutzgrundverordnung.html

Allerdings stellt diese Aufgabe Experten vor große Herausforderungen, da hierfür keine automatischen Methoden oder Techniken zur Verfügung stehen und somit die erforderlichen Tätigkeiten manuell durchgeführt werden müssen [GAE+16]. Da es sich hierbei aber um sehr komplexe und zeitaufwendige Tätigkeiten handelt, ist es für Domänenexperten nicht möglich, diese für gewachsene IT-Landschaften mit hunderten oder tausenden von Softwaresystemen durchzuführen.

So stehen Domänenexperten bei der Reduzierung unnötiger Variabilität von Technologiearchitekturen in gewachsenen IT-Landschaften konkret vor den folgenden *Problemen*:

1. **Variabilitätsbeziehungen sind nicht bestimmbar:** Für Experten ist es nahezu nicht möglich, explizite Variabilitätsbeziehungen für eine beliebige Anzahl von Technologiearchitekturen verschiedener Softwaresysteme zu bestimmen, da die komplexen Zusammenhänge von gemeinsam und unterschiedlich eingesetzten Komponenten manuell nicht erkennbar sind.

2. **Restrukturierungspotentiale sind nicht identifizierbar:** Es ist Experten kaum möglich, konkrete Restrukturierungspotentiale für beliebige Technologiearchitekturen zur Reduzierung ihrer Variabilität zu identifizieren, da der manuelle Aufwand für die Bestimmung von technisch machbaren Umstrukturierungen zur Eliminierung unnötiger Varianten zu hoch ist.

3. **Entscheidungsunterstützung ist nicht verfügbar:** Experten können keine objektiv nachvollziehbaren Restrukturierungsentscheidungen zur Variabilitätsreduzierung in Technologiearchitekturen treffen, da keine quantitative Entscheidungsgrundlage als Unterstützung zur Verfügung steht. Somit werden Entscheidungen über Restrukturierungen meist auf Basis subjektiver Einschätzungen getroffen.

1.2 Stand der Technik

Im Folgenden wird ein Überblick über den Stand der Technik im Kontext dieser Arbeit gegeben. Dabei werden verfügbare Ansätze aus der Wissenschaft berücksichtigt, welche sich um die Lösung ähnlicher Probleme bemühen, wie sie im Abschnitt 1.1 vorgestellt wurden.

Variabilitätsbestimmung: Zur Bestimmung von Variabilität in Softwarearchitekturen sind bereits einige Ansätze im Rahmen der *Software Product Line (SPL)*-Domäne entwickelt worden. So existieren beispielsweise Verfahren, welche die Variabilität von Source-Code-Artefakten bestimmen können (z.B. [ACM+11, SSS15, LLHE17]). Zudem gibt es auch Methoden, welche die Variabilität von Modellen identifizieren können (z.B. [NSC+07, ZHMP11, MZB+15]). Zwar wären einige dieser Ansätze generisch

genug, um sie auch für die Analyse der Variabilität von Modellen einer Technologiearchitektur zu verwenden (z.B. [ZHMP11, MZB+15]), jedoch berücksichtigen diese keine strukturellen Besonderheiten (z.B. Layer-Tier-Strukturen oder Kategoriensysteme für Komponenten). Somit erlauben die existierenden Ansätze aus der SPL-Domäne kein effizientes Variabilitätsmining für Technologiearchitekturen, weshalb die Variabilitätsbestimmung auf Ebene der Technologie auch immer noch als offenes Forschungsthema angesehen wird [GZW+17].

Darüber hinaus existieren auch Ansätze in der EA-Domäne, welche zur Bestimmung der Variabilität in Technologiearchitekturen beitragen. So erlauben verschiedene Methoden (z.B. [SM03, SWK13, LBMA14]) die Identifizierung der Heterogenität von eingesetzten Technologiekomponenten. Andere Verfahren ermöglichen die Analyse von EA-Modellen (z.B. [BJS13, SKR13, ABCB15]) und stellen Methoden bereit, um deren Diversität zu bestimmen. All diese Ansätze haben allerdings gemein, dass sie keine expliziten Variabilitätsbeziehungen zwischen den einzelnen Komponenten von analysierten Technologiearchitekturen bestimmen können. Insofern sind auch diese nicht zur Lösung unseres Problems geeignet.

Restrukturierungspotentiale: Zur Identifizierung von Restrukturierungspotentialen existieren eine Reihe von Ansätzen in der EA-Domäne. Zum einen schlagen diese Ansätze unterschiedliche Vorgehensmodelle als Rahmenwerk zur selbstständigen Herleitung konkreter Restrukturierungsmaßnahmen vor (z.B. [Dur08, BLS11, SM15]). Und zum anderen liefern viele dieser Arbeiten allgemeine Handlungsempfehlungen, welche zur Reduzierung der Heterogenität in Technologiearchitekturen beitragen (z.B. [Han11, FP14, SSJ14]). All diese Ansätze eignen sich jedoch nicht zur Lösung unseres Problems, da sie manuell durchgeführt werden müssen und damit im Kontext von gewachsenen IT-Landschaften nicht anwendbar sind.

Darüber hinaus gibt es auch automatisierte Verfahren, welche gegebene Technologiearchitekturen analysieren und Architekturempfehlungen ausgeben können (z.B. [BLNS12, Bus14, BSB+17]). Allerdings sind diese Ansätze nicht darauf ausgelegt, unnötige Variabilität in Technologiearchitekturen zu erkennen und entsprechende Restrukturierungsempfehlungen zu deren Reduzierung abzuleiten. Vielmehr stehen hierbei alternative Lösungsarchitekturen im Vordergrund, welche durch entsprechende Indikatoren bewertet und mit Hilfe von Technologieempfehlungen realisiert werden können. Somit liefern auch diese Arbeiten keinen geeigneten Lösungsansatz für unser beschriebenes Problem.

Entscheidungsunterstützung: Um Experten bei der Findung von Restrukturierungsentscheidungen zu unterstützen, sind Methoden erforderlich, welche die identifizierten Restrukturierungspotentiale bewerten und dadurch miteinander vergleichbar machen. Nur so lassen sich die verschiedenen Restrukturierungsmöglichkeiten gegeneinander abwägen und objektiv nachvollziehbare Entscheidungen treffen. In der Literatur werden hierfür verschiedene Ansätze vorgestellt, welche für die Bewertung von Unternehmens- beziehungsweise Technologiearchitekturen verwendet

werden können. Dabei stehen entweder Verbesserungen von Qualitätsmerkmalen einer Technologiearchitektur (z.B. [OJK+13, KSD14, EN16]) oder die allgemeine Unterstützung von Umstrukturierungen, beispielsweise durch eine Change-Impact-Analyse (z.B. [JPD09, LJH10, BZ18]), im Vordergrund. Da all diese Ansätze jedoch kaum automatisierte Methoden anbieten, ist weiterhin ein hoher manueller Aufwand erforderlich. Zudem stellen sie keine spezifischen Techniken zur Architekturbewertung mit dem Fokus auf die Variabilitätsreduzierung bereit. So lassen sich Restrukturierungspotentiale im Kontext von gewachsenen IT-Landschaften mit diesen Ansätzen nicht zielgerichtet bewertet.

Zusätzlich existieren weitere Ansätze, welche Methoden zur Entscheidungsunterstützung für Domänenexperten zur Verfügung stellen. Hierfür schlagen die jeweiligen Arbeiten entweder manuelle Unterstützungsmethoden vor, welche entsprechende Prozesse und Metamodelle beinhalten (z.B. [CvL10, NvP11, JSZ15]) oder semi-automatisierte Verfahren (z.B. [LZ13, PdP14]). Letztere unterstützen bei der Findung der bestmöglichen Architekturentscheidung auf Basis von verschiedenen Design-Alternativen. Hierfür verwenden sie entweder Entscheidungsmodelle oder Entscheidungsgraphen. Jedoch verfolgen auch diese Ansätze nicht das Ziel, die Variabilität in gewachsenen Technologiearchitekturen zu reduzieren. Aus diesem Grund fehlt bei ihnen die Berücksichtigung wesentlicher Aspekte für die Variabilitätsreduzierung (z.B. die Variabilitätsdifferenz oder die potentiellen Restrukturierungsaufwände), welche für die Entscheidungsfindung von Domänenexperten zur Verfügung stehen müssen. Insofern sind auch diese Ansätze nicht dazu geeignet, dass beschriebene Problem zu lösen.

1.3 Forschungsfragen

Mit Blick auf den aktuellen Stand der Technik lässt sich feststellen, dass es bisher keine geeigneten Verfahren gibt, welche Domänenexperten bei der Reduzierung unnötiger Variabilität von Technologiearchitekturen in gewachsenen IT-Landschaften unterstützen. Daher wird die folgende *zentrale Forschungsfrage* für die vorliegende Doktorarbeit aufgestellt:

Wie können Domänenexperten bei der Reduzierung unnötiger Variabilität von Technologiearchitekturen in großen, gewachsenen IT-Landschaften unterstützt werden?

Unter Berücksichtigung der im Abschnitt 1.1 beschriebenen Problemstellung können wir diese Frage in drei separate Forschungsfragen (FF) unterteilen:

FF1 – *Wie können explizite Variabilitätsbeziehungen zwischen individuellen Technologiearchitekturen identifiziert werden?*

Diese Forschungsfrage zielt darauf ab, einen (semi-)automatischen Ansatz zu entwickeln, der in der Lage ist, eine beliebige Anzahl von Technologiearchitekturen verschiedenster Softwaresysteme zu analysieren und so die expliziten Variabilitätsbeziehungen zwischen den verbauten Komponenten zu bestimmen.

FF2 – *Wie können konkrete Restrukturierungsempfehlungen zur Reduzierung unnötiger Variabilität bestimmt werden?*

Im Rahmen dieser Forschungsfrage soll ein (semi-)automatisches Verfahren entwickelt werden, welches unnötige Variabilität in den analysierten Technologiearchitekturen identifizieren und konkrete Restrukturierungsempfehlungen zur Reduzierung dieser unnötigen Variabilität ableiten kann.

FF3 – *Wie können Domänenexperten bei der Entscheidungsfindung zur Umsetzung von Restrukturierungsempfehlungen unterstützt werden?*

Ziel dieser Forschungsfrage ist es, einen (semi-)automatischen Ansatz zu entwickeln, der die identifizierten Restrukturierungsempfehlungen bewerten und deren Umsetzung mittels einer Soll-Architektur simulieren kann. Dies soll so erfolgen, dass Domänenexperten auf dieser Grundlage geeignete Restrukturierungsentscheidungen treffen können.

1.4 Beiträge und Aufbau der Arbeit

Im Rahmen dieser Arbeit werden drei eigene wissenschaftliche Beiträge vorgestellt, welche die definierten Forschungsfragen im Abschnitt 1.3 beantworten und sich damit zur Lösung der im Abschnitt 1.1 vorgestellten Problemstellung eignen. Konkret handelt es sich dabei um die folgenden Beiträge:

1. Ein Mining-Verfahren zur automatischen Bestimmung von Variabilität in Technologiearchitekturen, welches eine beliebige Anzahl von Modellen der Technologiearchitektur unterschiedlichster Softwaresysteme analysieren und dadurch alle enthaltenen Variabilitätsbeziehungen identifizieren kann (*FF1*).

2. Ein regelbasiertes Verfahren zur automatischen Ableitung von Restrukturierungsempfehlungen, welches unnötige Variabilität in den betrachteten Technologiearchitekturen identifizieren und geeignete Maßnahmen zur Reduzierung dieser Variabilität vorschlagen kann (*FF2*).

3. Ein Ansatz zur automatischen Bewertung und Simulation von abgeleiteten Restrukturierungsempfehlungen, welcher Experten bei der Findung von konkreten Restrukturierungsentscheidungen für die betrachteten Technologiearchitekturen unterstützen kann (*FF3*).

Diese drei Verfahren wurden in einen Gesamtansatz integriert, der Domänenexperten bei der Reduzierung unnötiger Variabilität von Technologiearchitekturen in großen, gewachsenen IT-Landschaften unterstützt. Die folgende Abbildung 1.1 stellt den Workflow für diesen Gesamtansatz grafisch dar.

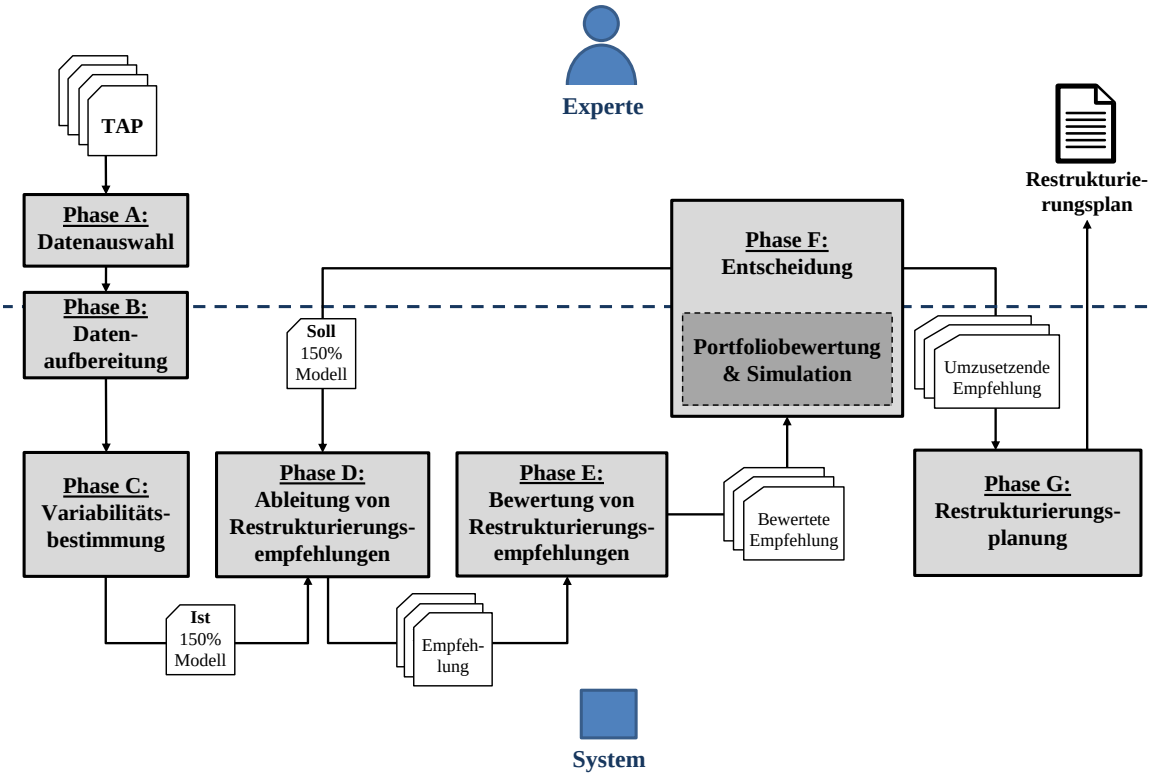


Abbildung 1.1: Workflow für den Gesamtansatz

Wie in Abbildung 1.1 zu erkennen ist, handelt es sich bei diesem Gesamtansatz um ein semi-automatisches Verfahren, welches manuelle (von *Experten* durchgeführte) und automatische (vom *System* durchgeführte) Bestandteile hat. Dieser Ansatz ist darauf ausgelegt, Domänenexperten bei der Entscheidungsfindung zur Reduzierung unnötiger Variabilität durch automatisierte Methoden zu unterstützen, erfordert aber an einigen Stellen das umfangreiche und meist nur implizit vorhandene Domänenwissen von Experten. Der Gesamtansatz besteht insgesamt aus den *sieben Phasen A–G*, welche nachfolgend kurz erläutert werden.

Phase A – Datenauswahl (manuell): In dieser ersten Phase wählt ein Domänenexperte die zu analysierenden Softwaresysteme anhand ihrer systemspezifischen Technologiearchitekturen, auch als *Technologiearchitekturplan (TAP)* bezeichnet, aus.

Phase B – Datenaufbereitung (semi-automatisch): In dieser zweiten Phase werden die ausgewählten TAPs mit Hilfe unterschiedlicher Methoden so aufbereitet, dass Fehler in den Daten, die während der manuellen Dokumentation irrtümlich eingebracht wurden, erkannt und ausgebessert werden. Dabei kann ein betrachteter Datenfehlertyp automatisiert und ein weiterer semi-automatisch mit Unterstützung von Domänenexperten bereinigt werden.

Phase C – Variabilitätsbestimmung (automatisch): In dieser dritten Phase erfolgt die Bestimmung der Variabilität für die ausgewählten TAPs. Als Ergebnis wird ein sogenanntes *150%-Modell* erzeugt, welches alle gemeinsam und unterschiedlich eingesetzten Komponenten zusammen mit ihren expliziten Variabilitätsbeziehungen beinhaltet. Da dieses Modell die Ist-Architektur der analysierten TAPs repräsentiert, wird es auch als *Ist-150%-Modell* bezeichnet.

Phase D – Ableitung von Restrukturierungsempfehlungen (automatisch): In dieser vierten Phase werden auf Grundlage des generierten Ist-150%-Modells Empfehlungen für konkrete Restrukturierungsmaßnahmen abgeleitet, welche zur Reduzierung von unnötiger Variabilität in den betrachteten TAPs beitragen. Hierfür werden zuerst Restrukturierungspotentiale aus einer businessorientierten Perspektive ermittelt und dann deren technische Machbarkeit aus einer technologieorientierten Sicht überprüft. Als Ergebnis werden anschließend konkrete Restrukturierungsempfehlungen für die analysierten TAPs erzeugt.

Phase E – Bewertung von Restrukturierungsempfehlungen (automatisch): In dieser fünften Phase wird jede einzelne der erzeugten Restrukturierungsempfehlungen im Hinblick auf ihr Potential zur Variabilitätsreduzierung bewertet. Außerdem werden im Rahmen dieser Bewertung ebenfalls die mit einer Restrukturierungsempfehlung verbundenen Aufwände geschätzt sowie potentiell abhängige Komponenten identifiziert, für die eine Anpassung nach erfolgter Restrukturierung notwendig ist, um die volle Funktionsfähigkeit des umstrukturierten Softwaresystems aufrechtzuerhalten.

Phase F – Entscheidung (semi-automatisch): In dieser sechsten Phase werden die konkreten Restrukturierungsentscheidungen durch Domänenexperten getroffen. Hierfür findet zuerst eine Bewertung des Gesamtportfolios aller Restrukturierungsempfehlungen auf Basis der Ergebnisse ihrer Einzelbewertungen (*Phase E*) statt, indem diese Ergebnisse verglichen und so lohnenswerte Restrukturierungsempfehlungen identifiziert werden. Anschließend kann die Entscheidungsfindung erfolgen, bei der die Ergebnisse der Einzel- und Portfoliobewertung den Experten als quantitative Grundlage zur Verfügung stehen. Zur weiteren Unterstützung können Domänenexperten beliebige Restrukturierungsempfehlungen für die Simulation auswählen, wodurch das resultierende *Soll-150%-Modell* automatisch generiert wird. Hierdurch lässt sich die Soll-Architektur der analysierten TAPs nach Umstrukturierung inspizieren. Erkennen Experten weiteres Potential zur Variabilitätsreduzierung in diesem Soll-150%-Modell, kann hierfür der Ansatz neu durchlaufen werden, indem das erzeugte Modell als Input für die *Phase D* verwendet wird. So können sich Experten schrittweise an ihre optimale Architekturlösung herantasten. Das Ergebnis dieser Phase stellen dann die Restrukturierungsempfehlungen dar, für die eine Entscheidung zur Umsetzung durch Domänenexperten getroffen wurde.

Phase G – Restrukturierungsplanung (automatisch): In dieser letzten Phase wird ein Restrukturierungsplan für alle zur Umsetzung vorgesehenen Restrukturierungsempfehlungen erstellt. Dabei werden konkrete Restrukturierungs- und Anpassungsmaßnahmen für jedes einzelne betroffene Softwaresystem erzeugt und in eine für die Umsetzung geeignete Reihenfolge gebracht. Der Restrukturierungsplan wird Domänenexperten anschließend zur Verfügung gestellt und kann als Grundlage für die Initiierung von entsprechenden Restrukturierungsprojekten verwendet werden.

Die genannten Phasen unseres Gesamtansatzes werden im Verlauf dieser Arbeit im Detail beschrieben. Den allgemeinen Aufbau der vorliegenden Arbeit zeigt die folgende Abbildung 1.2.

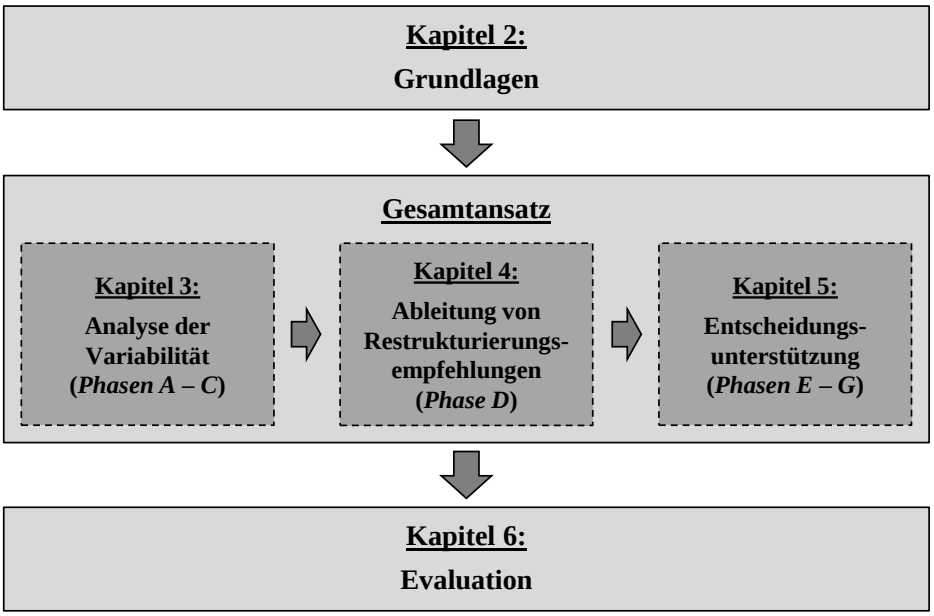


Abbildung 1.2: Aufbau der Arbeit

Im Kapitel 2 werden die Grundlagen für diese Arbeit gelegt. Anschließend wird der Gesamtansatz detailliert in den Kapiteln 3 bis 5 vorgestellt. Dabei beschreibt Kapitel 3 den Ansatz zur Analyse der Variabilität von systemspezifischen TAPs (*Phasen A–C*), Kapitel 4 den Ansatz zur Ableitung konkreter Restrukturierungsempfehlungen zur Reduzierung der unnötigen Variabilität in den analysierten TAPs (*Phase D*) und Kapitel 5 die Entscheidungsunterstützung für Domänenexperten in Form von automatisierten Methoden zur Bewertung und Simulation der identifizierten Restrukturierungsempfehlungen sowie zur anschließenden Restrukturierungsplanung (*Phasen E–G*). Abschließend wird dann im Kapitel 6 die Evaluation der beschriebenen Ansätze auf Basis von Fallstudien mit realen Architekturdaten aus der Industrie und Experteninterviews mit erfahrenen IT-Architekten unseres Industriepartners vorgestellt.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begrifflichkeiten und Konzepte vorgestellt, welche im Kontext der vorliegenden Arbeit relevant sind. Dazu findet zuerst eine Einführung in die Thematik der *Variabilität* (vgl. Abschnitt 2.1) und der *Technologiearchitekturen* (vgl. Abschnitt 2.2) statt, bevor das in dieser Arbeit eingesetzte Konzept der domänenspezifischen Sprachen (vgl. Abschnitt 2.3) erläutert wird.

2.1 Variabilität

Im folgenden wird der Begriff *Variabilität* eingeführt. Dazu erfolgt zuerst eine Einordnung und Begriffsbestimmung im Unterabschnitt 2.1.1, bevor die Modellierung von Variabilität im Unterabschnitt 2.1.2 dargestellt sowie die Techniken für ihre Realisierung im Unterabschnitt 2.1.3 und ihre Extraktion im Unterabschnitt 2.1.4 erläutert werden.

2.1.1 Einordnung und Begriffsbestimmung

Das Forschungsgebiet rund um die Thematik der Variabilität, welche auch als Software-Diversität bezeichnet wird [SRC+12], wurde vor allem im Kontext der *Software Product Line (SPL)*-Domäne geprägt [BCS10].

Definition 2.1: Software Product Line

Als *Software Product Line (SPL)* wird ein Set von software-intensiven Systemen verstanden, welche Gemeinsamkeiten in Form von Features aufweisen, die einen spezifischen Bedarf eines bestimmten Marktes befriedigen [Bos00, CN07]. Dabei bezeichnet ein *Feature* ein für Endnutzer sichtbares Merkmal eines Softwaresystems [KCH+90].

Solche SPLs können auch als Familie von Softwaresystemen verstanden werden [Sei17] und schaffen Vorteile im Rahmen der Softwareentwicklung durch die Wiederverwendung von sogenannten *Assets*, welche auch als Entwicklungs- oder Realisierungsartefakt bezeichnet werden. Hierdurch können beispielsweise die Entwicklungszeit und die damit verbundenen Entwicklungskosten reduziert sowie die Produktqualität erhöht werden [CN07].

Definition 2.2: Asset

Ein *Asset* ist ein Entwicklungs- bzw. Realisierungsartefakt, welches im Rahmen des Entwicklungsprozesses entsteht und Anforderungen, Architekturen, Code-Komponenten und Tests beinhaltet [PBv05].

Typischerweise wird ein Feature einer SPL durch eines oder mehrere Assets realisiert. So entstehen unterschiedliche Softwaresysteme innerhalb einer SPL durch die Wiederverwendung von gemeinsamen Assets in einer vorbestimmten Art und Weise [CN07]. Dabei werden die Features einer SPL dem sogenannten *Problem Space* und die entsprechenden Realisierungsartefakte dem *Solution Space* zugeordnet.

Definition 2.3: Problem Space und Solution Space

Der *Problem Space* beinhaltet konzeptionelle, variable Elemente (z.B. Features) einer SPL, welche relevant für die Softwareprodukte einer spezifischen Domäne sind und in Variabilitätsmodellen abgebildet werden. Im Vergleich dazu enthält der *Solution Space* wiederverwendbare Assets, also Realisierungsartefakte (z.B. Source-Code), für alle möglichen Softwaresysteme der Softwarefamilie einer betrachteten Domäne [CE00].

Die Abbildung 2.1 stellt den Zusammenhang von Problem und Solution Space dar und illustriert die Mechanismen zur *Realisierung* und *Extraktion* von *Variabilität* für Softwaresysteme einer SPL. Dies wird nachfolgend näher beschrieben.

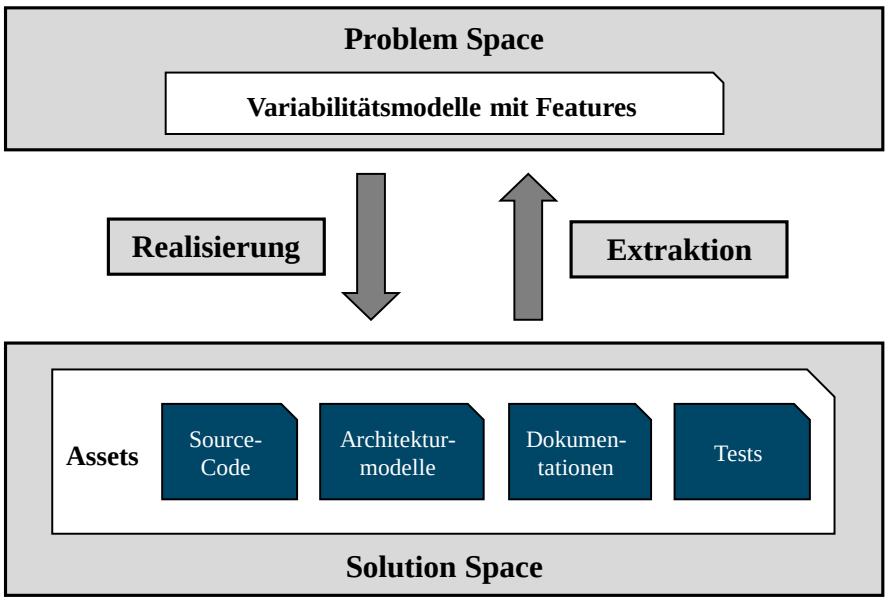


Abbildung 2.1: Variabilität im Kontext von Problem und Solution Space

Im Allgemeinen kann der Begriff *Variabilität* mit Hilfe der folgenden Definition beschrieben werden.

Definition 2.4: Variabilität

Unter *Variabilität* wird die Fähigkeit eines Softwaresystems oder eines Softwareartefaktes (z.B. einer Code-Komponente) verstanden, für einen spezifischen Kontext angepasst werden zu können. Diese Anpassung wird typischerweise vorausschauend geplant und kann beispielsweise durch Konfiguration, Individualisierung und Erweiterung erfolgen [GAWM11, GWT+14].

Gemeinsam und unterschiedlich eingesetzte Assets innerhalb einer Familie von Softwaresystemen bestimmen ihre Variabilität im Solution Space. Um die Wiederverwendung solcher Assets (auch in großem Umfang) zu ermöglichen, werden diese im Rahmen des *Software Product Line Engineering (SPLE)* gemanagt. Dabei kann SPLE als Paradigma zur Entwicklung von Softwaresystemen mit Hilfe von Plattformen (Sammlung von wiederverwendbaren Artefakten) und ihren individuellen Anpassungen beschrieben werden [PBv05]. Das Management von Variabilität ist eine der Kernaufgaben des SPLEs und hat das Ziel, Abhängigkeiten zwischen Varianten von Softwareartefakten zu steuern und deren Wiederverwendung im gesamten Lebenszyklus einer SPL zu unterstützen [BCS10]. Dadurch wird das Management von Gemeinsamkeiten und Unterschieden zwischen einzelnen Softwaresystemen, die Entwicklung von Varianten und Versionen, die Wiederverwendung von Artefakten in verschiedenen Softwareprodukten sowie die Unterstützung von Design-Entscheidungen ermöglicht [GAWM11].

Zur Modellierung der Variabilität im Problem Space wird das konkrete Wissen von Experten einer betrachteten Domäne benötigt, um konzeptionelle Element (z.B. Features) mit Hilfe von Variabilitätsmodellen beschreiben zu können. Auf dessen Basis lassen sich dann entsprechende Konfigurationen für spezifische Softwareprodukte erstellen. Solch eine *Konfiguration* ist eine gültige Teilmenge aller konfigurierbaren Features einer SPL auf konzeptioneller Ebene und bestimmt dadurch die Funktionalität eines einzelnen Softwaresystems [Sei17]. Somit werden konkrete Softwaresysteme einer SPL realisiert, indem die jeweiligen Assets im Solution Space ausgewählt werden, die zur Umsetzung der beabsichtigten Konfiguration aus dem Problem Space erforderlich sind. Wie in Abbildung 2.1 dargestellt ist, wird hierzu allerdings ein Ansatz benötigt, der das Mapping von Problem Space Variabilität und Solution Space Variabilität unterstützt. Zu diesem Zweck kann ein sogenannter Mechanismus für die *Realisierung* von Variabilität verwendet werden. Dieser erlaubt die Erstellung eines individuellen Softwareproduktes im Rahmen einer SPL durch die Zusammenstellung von Realisierungsartefakten auf Basis einer konzeptionellen Konfiguration für spezifische Anforderungen [SRC+12]. Ist hingegen die nachträgliche Identifizierung eines Variabilitätsmodells für vorhandene Realisierungsartefakte einer SPL gewünscht, so wird ein Mechanismus zur *Extraktion*

von Variabilität benötigt. Solche Ansätze dienen der Bestimmung von inhärenten Variabilitätsinformationen für ein konkretes Set von Realisierungsartefakten, um daraus konzeptionelle Variabilitätsmodelle erstellen zu können [Kru01, MZB+15].

Wird der Begriff Variabilität etwas genauer betrachtet, so lässt sich dieser aus unterschiedlichen Perspektiven beschreiben. Dabei kann eine Unterscheidung in den drei Dimensionen *Raum und Zeit*, *Intern und Extern* sowie *beabsichtigt und unbeabsichtigt* erfolgen.

Definition 2.5: Variabilität in Raum und Zeit

Als *Variabilität im Raum* wird die Existenz eines bestimmten Artefaktes in verschiedenen Ausführungen (Varianten) verstanden, welche alle zur selben Zeit gültig sind. Dagegen beschreibt die *Variabilität in der Zeit* die Existenz verschiedener Versionen eines Artefaktes, welche zu unterschiedlichen Zeiten Gültigkeit besitzen [PBv05].

Mit der Variabilität im Raum können verschiedene Ausprägungen eines Softwaresystems oder eines Artefaktes beschrieben werden, die auf Basis von unterschiedlichen Konfigurationen, welche Individualisierungen und Erweiterungen beinhalten können, entstehen. So kann durch eine spezifische Konfiguration von Features, auch *Produktvariante* genannt [LLHE17], ein konkretes Softwareprodukt einer SPL erstellt werden. Im Fokus der Variabilität in der Zeit liegen die verschiedenen Versionen eines Softwaresystems oder Artefaktes. Diese entstehen durch Weiterentwicklung, auch *Software Evolution* genannt, um beispielsweise neue Anforderungen zu erfüllen oder enthaltene Fehler zu korrigieren [Leh80].

Definition 2.6: Interne und externe Variabilität

Als *intern* wird die Variabilität von Artefakten bezeichnet, wenn sie von außen durch Nutzer nicht erkennbar ist. Im Gegensatz dazu beschreibt die *externe Variabilität* die durch Nutzer von außen sichtbare Variabilität [PBv05].

Die Unterscheidung zwischen interner und externer Variabilität zielt vor allem auf die Kunden beziehungsweise Nutzer einer SPL ab. So können variable Elemente, welche von außen nicht sichtbar sind, lediglich von Entwicklern der Systeme einer Softwarefamilie berücksichtigt werden. Für die Konfiguration von diversitären Produkten solch einer Softwarefamilie ist es hingegen unerlässlich, die Produkte nach variablen Artefakten zu schneiden, die auch nach außen für deren Nutzer sichtbar sind.

Neben den bereits beschriebenen Dimensionen kann Variabilität auch *im Kontext* betrachtet und dabei zwischen *beabsichtigter* und *unbeabsichtigter* Variabilität unterschieden werden [GZW+17].

Definition 2.7: Beabsichtigte und unbeabsichtigte Variabilität

Die *beabsichtigte Variabilität* bezeichnet die geplanten und gewünschten variablen Artefakte eines Softwaresystems, mit deren Hilfe beispielsweise unterschiedliche, fachliche Anforderungen realisiert werden können. Hingegen beschreibt die *unbeabsichtigte Variabilität* ungeplante und ungewollte variable Artefakte, welche als Nebeneffekt von Entwicklungstätigkeiten entstehen, die nicht das Ziel haben, Variabilität zu erzeugen [GZW+17].

Somit steht hinter der beabsichtigten Variabilität ein geplantes Vorgehen für die Entwicklung von variablen Artefakten, beispielsweise durch die Abbildung von zusätzlichen Features im Variabilitätsmodell einer SPL. Im Gegensatz dazu werden diversitäre Elemente, die der unbeabsichtigten Variabilität zugeordnet werden können, irrtümlicherweise erzeugt. Dies kann zum Beispiel bei der Entwicklung einer zusätzlichen Softwarekomponente durch Kommunikationsschwierigkeiten im Entwicklungsteam oder durch Nebeneffekte bei der Erweiterung eines Softwaresystems um weitere Funktionen verursacht werden [Bro87].

2.1.2 Modellierung von Variabilität

Das Wissen über die Variabilität von SPLs, auf dessen Basis spezifische Varianten und Versionen von Softwaresystemen konfiguriert werden können, ist häufig nur implizit in den Köpfen von Domänenexperten vorhanden [SRC+12]. Um dieses Wissen auch für andere verfügbar zu machen, muss es explizit beschrieben und verankert werden. Hierfür eignen sich Variabilitätsmodelle, welche alle Gemeinsamkeiten und Unterschiede von Softwaresystemen durch die Beschreibung von Varianten ihrer Artefakte definieren und um organisations- sowie domänenspezifische Abhängigkeiten ergänzen. Der damit verbundene Prozess zur Definition und Dokumentation der Variabilität einer SPL wird als *Variabilitätsmodellierung* bezeichnet [Cza10].

In der Literatur finden sich unterschiedliche Modelle zu Abbildung von Variabilität, beispielsweise das *Feature Model* [KCH+90, CE00], Entscheidungsmodelle [MA02], das *Orthogonal Variability Model (OVM)* [PBv05] und die Spezifikation mit Hilfe der *Common Variability Language (CVL)* [HMPO+08]. Dabei ist das Feature Model der wohl bekannteste und am weitesten verbreitete Ansatz zur Modellierung von Variabilität innerhalb einer SPL [SRC+12]. Daher wird dieses im folgenden kurz vorgestellt.

Feature Model nach Kang et al.

Mit Hilfe eines Feature Models werden die nach außen sichtbaren Merkmale einer SPL, welche als Features bezeichnet werden (vgl. Definition 2.1), betrachtet. Solche Merkmale werden mit Hilfe einer hierarchischen Struktur abgebildet, um damit die gemeinsam und unterschiedlich verwendeten Features einer SPL zu beschreiben.

Hierfür wird jede hierarchische Assoziation zwischen einem Eltern- und einem Kindelement durch eine konkrete *Variabilitätsbeziehung* definiert [KCH+90, CE00].

Definition 2.8: Variabilitätsbeziehung

Eine *Variabilitätsbeziehung*, auch als Variabilitätsabhängigkeit bezeichnet, definiert eine Assoziation zwischen Features, die durch eine Eltern-Kind-Beziehung verbunden sind. Solch eine Beziehung kann vom Typ *verpflichtend*, *optional*, *OR* oder *alternativ* sein [KCH+90, CE00].

Ein verpflichtendes Feature wird in allen Varianten einer Systemfamilie implementiert. Zudem kann solch ein System entweder über ein oder kein optionales Feature sowie über eines oder mehrere Features einer OR-Gruppe verfügen. Darüber hinaus bildet eine Menge von alternativen Features ein Set von Auswahlmöglichkeiten, wobei je genau eine Alternative aus diesem Set pro System implementiert werden muss [KCH+90, CE00].

Für Features, die nicht über eine Eltern-Kind-Beziehung miteinander verbunden sind, können ebenfalls beschränkende Assoziationen in Form von *Cross-Tree-Constraints* beschrieben werden.

Definition 2.9: Cross-Tree-Constraints

Ein *Cross-Tree-Constraint* beschreibt eine Beschränkung für die Verwendung von zwei oder mehreren Features innerhalb einer Konfiguration für ein Softwaresystem, typischerweise durch Inklusion oder Exklusion [SRC+12].

Solche Cross-Tree-Constraints werden häufig durch aussagenlogische Formeln mit Hilfe von logischen Operatoren (z.B. $\wedge$ (und), $\vee$ (oder), $\neg$ (nicht)) definiert [Bat05, SWS16]. Dabei bezeichnet eine Inklusion `A implies B`, dass Feature A immer mit Feature B verwendet werden muss und eine Exklusion `A excludes B`, dass Feature A niemals zusammen mit Feature B implementiert werden kann.

Zur graphischen Repräsentation eines Feature Models werden sogenannte *Feature Diagramme* erstellt, welche hierarchisch strukturierten Features mit ihren Variabilitätsbeziehungen in geeigneter Art und Weise darstellen [KCH+90, CE00]. Die folgende Abbildung 2.2 zeigt ein einfaches Feature Diagramm für ein Fahrzeug. Da die Struktur eines Fahrzeuges naturgemäß sehr komplex ist, werden hier nur einige ausgewählte Features gezeigt. Wie hier zu sehen ist, besteht das beispielhafte Fahrzeug aus den verpflichtenden Features *Fahrertür*, *Beifahrertür*, *Motor* und *Radio* sowie dem optionalen Feature *Schiebedach*. Dem Motor sind allerdings die zwei Kindelemente *Benzin* und *Diesel* zugeordnet, welche zwei Alternativen darstellen. Somit muss für jede Konfiguration entweder ein Benzin- oder ein Dieselmotor ausgewählt werden. Ähnliches gilt für das Radio-Feature, welches eine OR-Gruppe mit

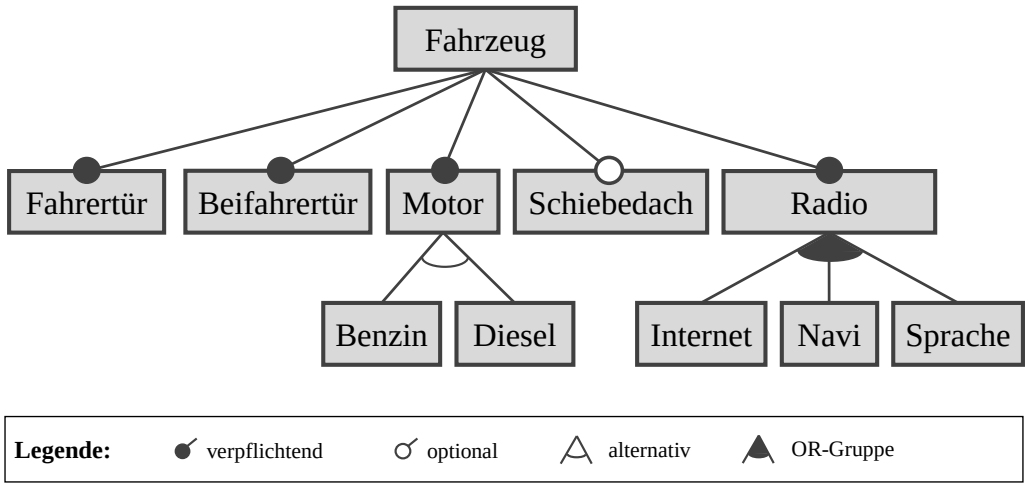


Abbildung 2.2: Beispielhaftes Feature Diagramm für ein Fahrzeug (Ausschnitt)

den Kindelementen *Internet*, *Navi* und *Sprache* beschreibt. Von diesen Features kann eines oder mehrere für eine spezifische Radio-Konfiguration ausgewählt werden.

Liegt zudem eine beispielhafte Beschränkung für die beiden Features *Schiebedach* und *Sprache* vor, sodass bei Einsatz eines Schiebedaches keine Verwendung des Sprach-Features möglich ist, lässt sich dies über die folgende aussagenlogische Formel abbilden: `Schiebedach excludes Sprache` $\equiv \neg(Schiebedach \wedge Sprache)$

2.1.3 Realisierung von Variabilität

Der Problem Space beinhaltet das beschriebene Variabilitätsmodell einer SPL. Um auf dieser Basis eine konzeptionelle Konfiguration für ein spezifisches Softwareprodukt in eine konkrete Realisierung eines Softwaresystems mit Hilfe von variablen Assets des entsprechenden Solution Spaces zu überführen, werden Mechanismen zur Realisierung dieser Variabilität benötigt. Hierbei kann zwischen drei Arten von Realisierungsansätzen unterschieden werden: *annotativ*, *kompositionell* und *transformierend* [SRC+12]. Während der annotative Ansatz auf einem einzelnen Modell basiert und Variabilität durch Annotationen an diesem Modell beschreibt, assoziiert der kompositionelle Ansatz Fragmente aus solch einem Modell mit Produkt Features, sodass diese für eine bestimmte Feature-Konfiguration zusammengesetzt werden können. Beim transformierenden Ansatz hingegen wird Variabilität durch die Transformation eines Modells der Basisvariante beschrieben. Im folgenden werden der annotative und der transformierende Mechanismus näher erläutert, da diese beiden Ansätze für die vorliegende Arbeit relevant sind.

Annotativer Mechanismus zur Realisierung von Variabilität

Der *annotative Mechanismus* zur Realisierung von Variabilität basiert auf der Idee, ein einzelnes Modell zu erstellen, welches alle variablen Artefakte einer SPL

beinhaltet. Solch ein Modell wird auch als überlagerte Variante oder *150%-Modell* bezeichnet [CA05, SRC+12].

Definition 2.10: 150%-Modell

Ein *150%-Modell* ist eine monolithische Repräsentation aller Varianten einer SPL innerhalb des Solution Spaces in Form eines einzelnen Modells [SRC+12].

Durch Annotationen an solch einem 150%-Modell kann für jede Produktvariante definiert werden, welche Teile von diesem Modell zu entfernen sind, um daraus ein konkretes Softwareprodukt für eine SPL ableiten zu können. Dabei können Annotationen beispielsweise durch *UML-Stereotypen* [Gom05] oder *Presence Conditions* [CA05] umgesetzt werden [SRC+12]. Der Mechanismus zur Erstellung konkreter Produktvarianten mit Hilfe eines annotierten 150%-Modells wird auch als *Prozess zur Variantenableitung* bezeichnet [Sei17]. Dieser Prozess ist beispielhaft in Abbildung 2.3 dargestellt und basiert dabei auf einem 150%-Modell mit den exemplarischen Annotationen `V1` und `V2`, welche die zugehörigen Artefakte für die Produktvarianten V1 und V2 beschreiben.

Wie in Abbildung 2.3 zu erkennen ist, findet zur Ableitung einer konkreten *Produktvariante V1* zunächst ein Mapping der Elemente einer ausgewählten konzeptionellen *Konfiguration V1* auf die entsprechenden Realisierungsartefakte im Solution Space statt. Erst danach kann die konkrete Systemvariante erzeugt werden, indem jene Artefakte aus dem 150%-Modell entfernt werden, welche auf Basis der enthaltenen Annotationen nicht für die entsprechende Produktvariante V1 benötigt werden. Im betrachteten Fall werden also die beiden Artefakte mit der Annotation `V2` entfernt.

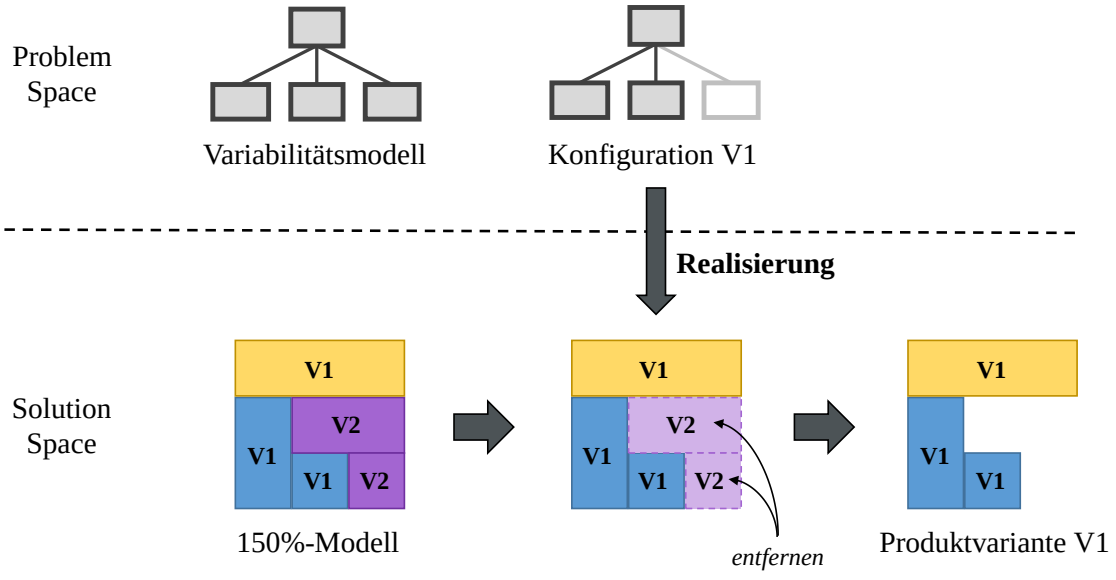


Abbildung 2.3: Annotativer Mechanismus zur Realisierung von Variabilität

Transformierender Mechanismus zur Realisierung von Variabilität

Ein weiterer Ansatz zur Realisierung von Variabilität im Solution Space ist der sogenannte *transformierende Ansatz*, mit dem konkrete Systemvarianten einer SPL durch Modelltransformationen erzeugt werden können [SRC+12]. Ein für diese Arbeit relevanter Mechanismus zur Transformation von Modellen ist die *Delta Modellierung* [SBB+10, SD10, CHS10, Sch10].

Im Rahmen der Delta Modellierung wird zuerst eine initiale Basisvariante, auch Kern-Modell genannt, für eine SPL erstellt. Diese berücksichtigt die Variabilität des zugrunde liegenden Feature Models und entspricht einer validen Konfiguration für ein Softwareprodukt aus dieser SPL [Sch10]. Solch eine Basisvariante kann dann mit Hilfe von *Deltaoperationen* zu einer neuen Produktvariante transformiert werden, indem Elemente *hinzugefügt*, *entfernt* oder *modifiziert* werden. Zu diesem Zweck wird im Rahmen der *Delta-orientierten Programmierung* eine auf die Zielsprache (z.B. *Java*) ausgerichtete Delta-Sprache (z.B. *DeltaJava*) verwendet, mit der spezifische Deltaoperationen für konkrete SPLs definiert werden können [SBB+10]. Zur Unterstützung der Entwicklung solcher Delta-Sprachen lassen sich entsprechende Frameworks (z.B. *DeltaEcore* [SSA14]) verwenden. Zudem können atomare Deltaoperationen zusammengefasst und zu sogenannten *Delta Modulen* (Kurzform *Deltas*) gruppiert werden, um dadurch zusammenhängende Änderungsoperationen zu bündeln. Somit kann eine konkrete Produktvariante erzeugt werden, indem die initiale Basisvariante durch Anwendung der entsprechenden Delta-Module transformiert wird. Die folgende Abbildung 2.4 stellt diesen Ansatz schematisch dar.

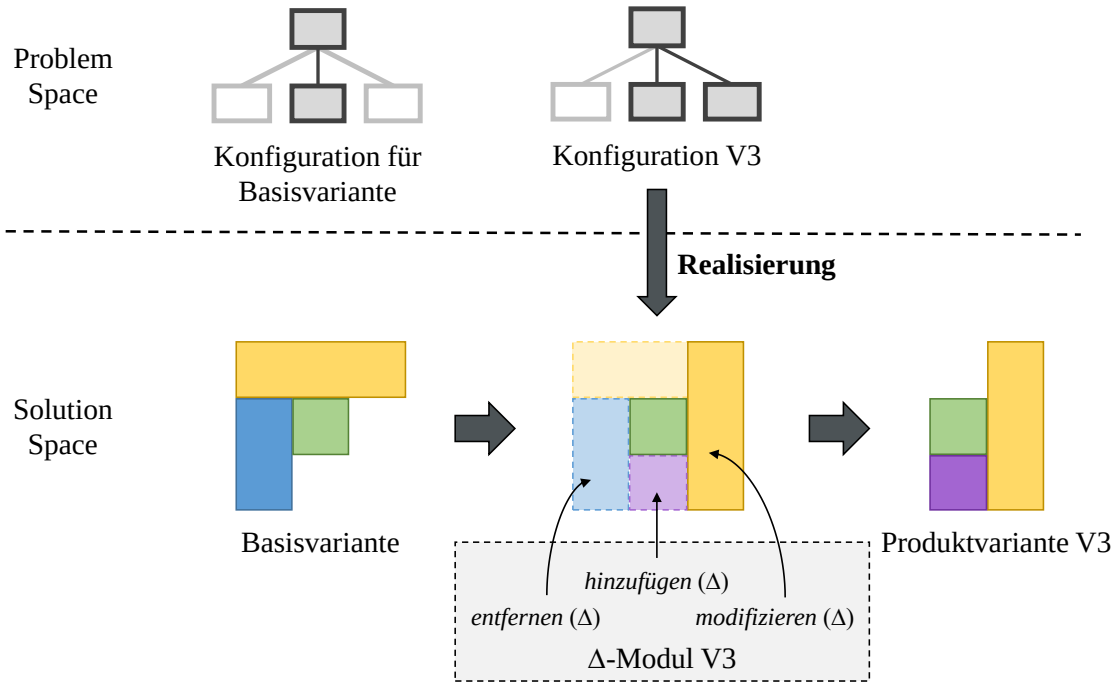


Abbildung 2.4: Transformierender Mechanismus zur Realisierung von Variabilität

Abbildung 2.4 zeigt die Delta Modellierung als transformierenden Ansatz zur Realisierung von Variabilität auf Grundlage einer Basisvariante. Solch ein valides Basismodell kann durch Auswahl einer spezifischen Konfiguration des konzeptionellen Variabilitätsmodells erzeugt werden. Durch die Anwendung des *Delta-Moduls V3* (Δ-Modul), bestehend aus den Deltaoperationen (Δ) „*Entfernen*“, „*Hinzufügen*“ und „*Modifizieren*“, kann das Basismodell transformiert und die konkrete *Produktvariante V3* abgeleitet werden.

2.1.4 Extraktion von Variabilität

Für die Realisierung von Variabilität steht ein geeignetes Variabilitätsmodell zur Verfügung, um auf dessen Basis konkrete Softwaresystem-Varianten erstellen zu können. Ist solch ein explizites Wissen über die existierende Variabilität innerhalb einer SPL nicht vorhanden, so ist für beteiligte Experten und Entwickler nicht unmittelbar erkennbar, welche Teile in betrachteten Produktvarianten gleich oder unterschiedlich sind. Somit ist ein Management von wiederverwendbaren Artefakten nicht möglich, was zu einem steigenden Aufwand für Wartung und Weiterentwicklung der Produktvarianten solcher SPLs führt.

Aus diesem Grund sind Ansätze zur Extraktion von Variabilität entwickelt worden, welche entweder manuelle Vorgehensweisen (z.B. [RDGB08]) oder automatisierte Verfahren (z.B. [Kru01, MZB+15, LLHE17]) vorschlagen, um die inhärente Variabilität innerhalb eines Sets von Produktvarianten nachträglich bestimmen zu können. Solche Verfahren werden auch als *Variabilitätsmining* [KDO14] oder *Family Mining* [WSSS16] bezeichnet und stellen Reverse-Engineering-Ansätze dar, welche die Bestimmung von Variabilitätsinformationen in verwandten Softwaresystemen ermöglichen. Hierfür können verschiedene Realisierungsartefakte aus dem Solution Space der jeweiligen Softwareprodukte berücksichtigt und analysiert werden, wie beispielsweise *Source-Code* [LLHE17] oder *State-Chart-Modelle* [WSS16]. Ein für diese Arbeit relevantes Verfahren zur automatisierten Extraktion von Variabilität ist das Variabilitätsmining nach Wille et al. [WSSS16, WRSS17, Wil18], welches im folgenden kurz vorgestellt wird.

Im Rahmen des Mining-Verfahrens nach Wille et al. werden die drei Phasen *Compare*, *Match* und *Merge* zur Bestimmung der Variabilität von Realisierungsartefakten durchgeführt. In der Compare-Phase werden zuerst alle Artefakte in Form von Modellen miteinander verglichen, um ihre Variabilitätsbeziehungen (vgl. Definition 2.8) zu identifizieren. Dabei erfolgt der Vergleich auf Ebene einzelner Modellelemente, um deren Ähnlichkeit mit Hilfe einer geeigneten Metrik zu bestimmen. In der folgenden Match-Phase werden anschließend die zuvor identifizierten Variabilitätsbeziehungen im Hinblick auf Mehrdeutigkeiten untersucht. Diese können beispielsweise entstehen, wenn sich mehrere Elemente eines Modells auf das gleiche Element eines anderen Modells beziehen. Sind solche Mehrdeutigkeiten vorhanden, werden sie durch die Bestimmung eindeutiger Beziehungen aufgelöst. In der abschließenden Merge-Phase werden dann alle Modellelemente aus den verglichenen Modellvarianten in ein

gemeinsames 150%-Modell (vgl. Definition 2.10) zusammengeführt. Dieses wird anschließend mit entsprechenden Annotationen versehen, welche die expliziten Variabilitätsbeziehungen zwischen den analysierten Modellelementen sowie ihre enthaltenen Varianten darstellen.

In Abbildung 2.5 ist das vorgestellte Mining-Verfahren zur Extraktion von Variabilität im Solution Space nach Wille et al. [WSSS16, WRSS17, Wil18] schematisch dargestellt. Mit Hilfe dieses dreistufigen Mining-Verfahrens können alle gemeinsam und unterschiedlich eingesetzten Artefakte sowie deren spezifische Variabilitätsbeziehungen für die drei verschiedenen *Produktvarianten V1–V3* identifiziert werden. Darauf aufbauend lässt sich dann das entsprechende 150%-Modell erstellen, welches alle *verpflichtenden (V)*, *alternativen (A)* und *optionalen (O)* Artefakte der jeweiligen Produktvarianten beinhaltet. Durch entsprechende Annotationen im 150%-Modell können die Variabilitätsbeziehungen der jeweiligen Artefakte gespeichert und die ursprünglichen Produktvarianten zugeordnet werden.

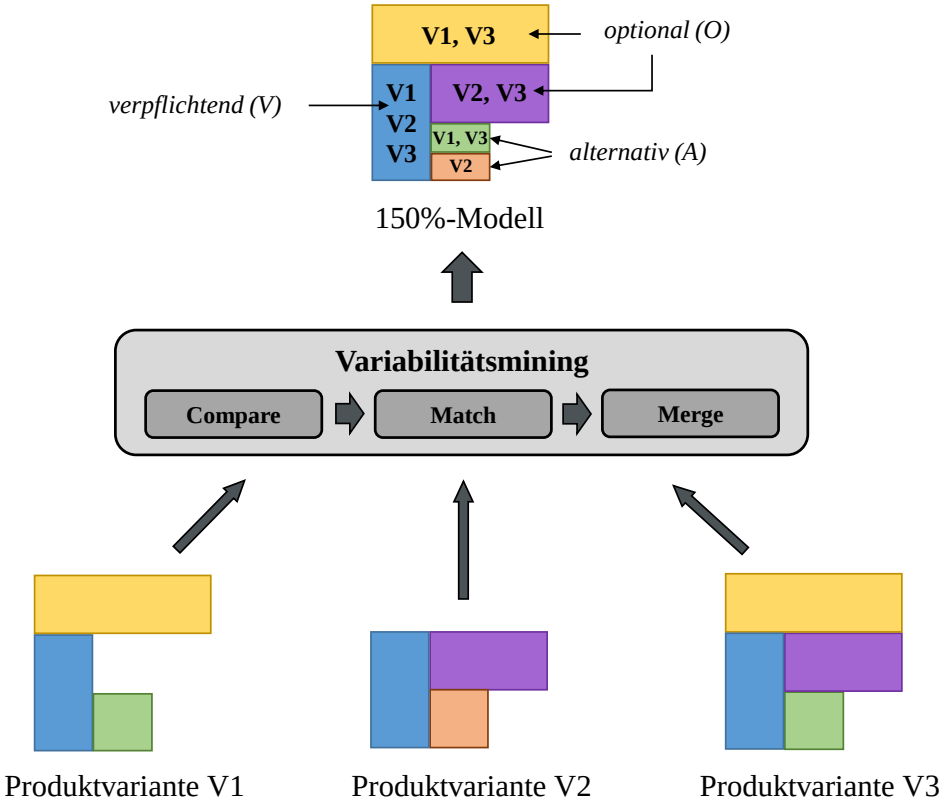


Abbildung 2.5: Mining-Verfahren zur Extraktion von Variabilität im Solution Space nach Wille et al.

2.2 Technologiearchitektur

Im folgenden Abschnitt werden die Grundlagen zur Thematik der *Technologiearchitektur* beschrieben. Dazu erfolgt zunächst eine Einordnung und Begriffsbestimmung im Unterabschnitt 2.2.1 und anschließend eine Beschreibung der Modellierungskonzepte für Technologiearchitekturen im Unterabschnitt 2.2.2.

2.2.1 Einordnung und Begriffsbestimmung

Der Begriff *Architektur* wird durch die ISO/IEC/IEEE Norm 24010-2011 definiert als die fundamentale Organisation eines Systems, welches durch seine Komponenten und den Relationen zwischen ihnen beschrieben ist [ISO11]. Im Kontext eines Unternehmens, beschreibt die *Unternehmensarchitektur*, auch *Enterprise Architecture (EA)* genannt, die fundamentale Organisation aller relevanten Aspekte eines Unternehmens [ASML12]. Daraus lässt sich die folgende Definition ableiten.

Definition 2.11: Unternehmensarchitektur

Als *Unternehmensarchitektur* oder *Enterprise Architecture (EA)* kann die fundamentale Konzeption einer gesamten Organisation verstanden werden, die durch ihre Elemente, deren Beziehungen untereinander und den Prinzipien zu ihrer Entwicklung und Evolution geprägt ist [Sch16].

Solche Elemente und Beziehungen beschreiben die relevanten Aspekte einer Organisation, wie beispielsweise die Aufbau- und Ablauforganisation, Fachfunktionen und -prozesse, Daten und Informationen sowie Applikationen und die technologische Infrastruktur [ASML12]. Zur Einordnung in fachliche Domänen und zur besseren Unterscheidung dieser Bestandteile, wird eine Unternehmensarchitektur in verschiedene Architekturebenen unterteilt. Hierfür finden sich in der Literatur unterschiedliche Ansätze, welche drei (z.B. [Kel12, EHH+08]), vier (z.B. [The11, SWG13]) oder gar fünf (z.B. [WF06, Der09]) Architekturebenen für die Konzeption einer EA vorschlagen. In der Praxis wird häufig ein 4-Schichten-Modell zur Darstellung der EA nach dem *The Open Group Architecture Framework (TOGAF)* [The11] bevorzugt[1], welches die Ebenen *Geschäftsarchitektur*, *Applikationsarchitektur*, *Datenarchitektur* sowie *Technologiearchitektur* definiert.

Eine solche Unternehmensarchitektur nach TOGAF ist schematisch in Abbildung 2.6 dargestellt. Dabei fokussiert die oberste Ebene der Geschäftsarchitektur auf organisationale Aspekte wie Strategien, Funktionen und Prozesse. Die darunter liegende Ebene der Applikationsarchitektur beschäftigt sich mit den Softwaresystemen im Unternehmen und ihren Interaktionen. Im Rahmen der Datenarchitektur werden

[1] Laut dem *IT Standardization Report 2015* der TU München ist TOGAF der am häufigsten eingesetzte Standard zur Darstellung von Unternehmensarchitekturen [SGM15]

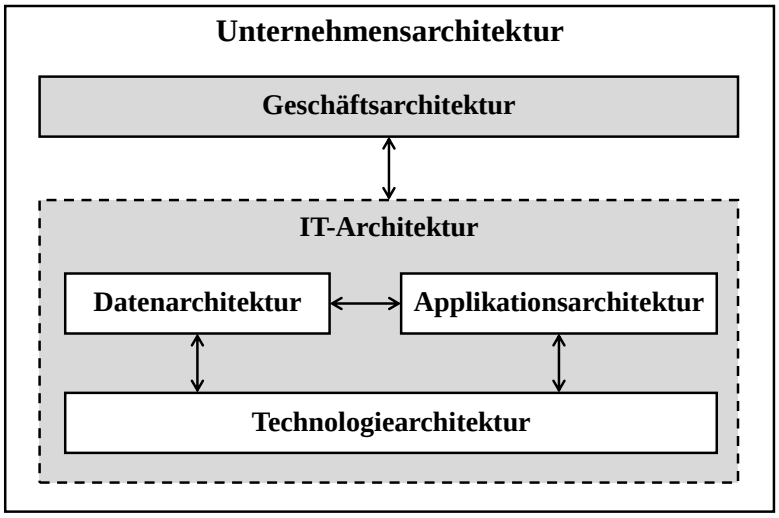


Abbildung 2.6: Ebenen einer Unternehmensarchitektur nach [The11, SWG13]

die Strukturen der von den Applikationen verarbeiteten Daten und Informationen betrachtet. Das Fundament für diese drei Architekturebenen bildet die Technologiearchitektur, da sie als unterste Ebene einer Unternehmensarchitektur die technische Infrastruktur für die darüber liegenden Ebenen bereit stellt. Somit beinhaltet eine EA zwei grundsätzliche Perspektiven: Eine businessorientierte Sicht auf die Geschäftsarchitektur sowie eine technologieorientierte Sicht auf die *IT-Architektur*, welche die drei Ebenen Daten-, Applikations- und Technologiearchitektur beinhaltet [Ros03a] und dadurch eine strukturierte Abstraktion der Softwaresysteme im Unternehmen erlaubt [Der09]. Im Rahmen dieser Arbeit liegt der Fokus auf der untersten Ebene solch einer EA, also auf der Technologiearchitektur, welche wie folgt definiert werden kann.

Definition 2.12: Technologiearchitektur

Als *Technologiearchitektur (TA)*, auch IT-Basisinfrastruktur genannt, wird „die Menge aller Hardware- und systemnahen Softwarekomponenten verstanden, die die Laufzeit- und Managementumgebung für Entwicklung, Test und Produktion von Informationssystemen bilden“ [Der09].

Demnach repräsentiert eine TA die fundamentale Organisation der Computerinfrastruktur eines Unternehmens, welche die notwendige Basis für den Betrieb von fachlichen Applikationen und die Verarbeitung der dafür erforderlichen Daten bereit stellt. Hierfür werden verschiedenste Komponenten auf Hardwareebene (z.B. Serversysteme und Netzwerkknoten) sowie auf Softwareebene (z.B. Betriebssysteme und Applikationsserver) für die Nutzung durch die Softwaresysteme der darüber

liegenden Applikationsarchitektur zur Verfügung gestellt [ASML12, WF06]. So beinhaltet eine TA alle technologischen Komponenten für die IT-Infrastruktur, Middleware, Netzwerke, Kommunikation und Datenverarbeitung [The11]. In diesem Kontext definieren wir den Begriff *Komponente* wie folgt.

Definition 2.13: Komponente

Eine *Komponente* ist ein Hardware- oder Softwareelement auf Ebene der Technologiearchitektur und stellt ein spezifisches Technologieprodukt dar.

Dabei lassen sich *logische* und *physische Komponenten* bzw. *Elemente* voneinander unterscheiden [The11].

Definition 2.14: Logische und physische Elemente

Ein *logisches Element* stellt eine Klasse von technologischen Komponenten dar, welche unabhängig von konkreten Technologien ist und diese kapselt. Ein *physisches Element* ist eine konkrete Technologie und stellt damit eine Instanz ihres logischen Elementes dar [The11].

Physische Elemente sind konkrete, einsetzbare Technologieprodukte, wie beispielsweise die Datenbanksysteme `MySQL 5.6` und `MySQL 5.7`. Logische Elemente kapseln diese in einer Klasse (z.B. `MySQL`) und beschreiben somit abstrakte Technologieprodukte, welche nicht direkt in einer Technologiearchitektur verwendet werden können. Der allgemeine Begriff Komponente (oder Element) berücksichtigt beide Typen.

2.2.2 Modellierung von Technologiearchitekturen

Im Rahmen der Modellierung von Technologiearchitekturen können zwei unterschiedliche Betrachtungsebenen unterschieden werden [Sch09]. Die erste Ebene wird als *Makroarchitektur* bezeichnet und berücksichtigt eine ganzheitliche Betrachtung der Technologiearchitektur im gesamten Unternehmen. Die zweite Ebene der *Mikroarchitektur* beschreibt einen fokussierten Blick auf die Technologiearchitektur eines einzelnen Softwaresystems. So lässt sich die Mikroarchitektur auch als Ausschnitt einer ganzheitlichen Makroarchitektur für ein spezifisches System beschreiben, was auch als *Sicht* oder *View* solch einer Architektur verstanden werden kann [ISO11].

Solch eine systemspezifische Sicht auf die Technologiearchitektur kann auch als *Komponentensicht* bezeichnet werden, welche die logische und funktionale Struktur der Architektur mit allen wesentlichen Komponenten in ihrer jeweiligen Zusammensetzung darstellt [LRR03]. Diese Sicht spielt für die vorliegende Arbeit eine wesentliche Rolle, da sie eine gute Basis für die vergleichende Analyse von individuellen Technologiearchitekturen darstellt und dadurch die Identifizierung von

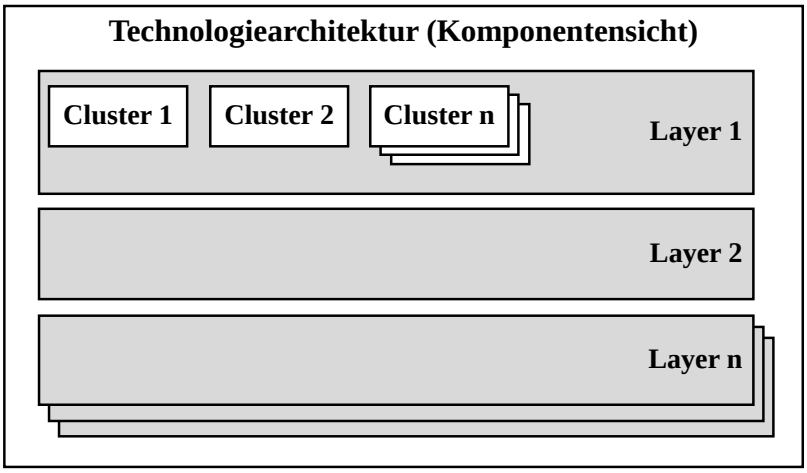


Abbildung 2.7: Strukturierung einer Technologiearchitektur nach [Liu02, Dur08]

Variabilität ermöglicht. Die systemspezifische Struktur und Zusammensetzung von Komponenten lässt sich dafür mit Hilfe eines Schichtenmodells abbilden, welches durch eine zweistufige Ordnung mit *Layern* und *Clustern* definiert ist [Liu02, Dur08]. Dieses Schichtenmodell ist in Abbildung 2.7 dargestellt.

Solch ein Schichtenmodell bildet einen funktionalen Ordnungsrahmen, mit dessen Hilfe technologische Komponenten auf Basis ihrer funktionalen Zugehörigkeit in Layern und Clustern eingeordnet werden können. So kann eine funktionale Abstraktion von einzelnen Technologien erfolgen [Dur08]. In diesem Zusammenhang wird der Begriff *Layer* wie folgt definiert.

Definition 2.15: Layer

Ein *Layer* ist eine horizontale Schicht innerhalb einer Technologiearchitektur und gliedert Cluster nach ihrer übergeordneten Funktion [Dur08].

Solche Layer werden nach einem abstrahierten ISO/OSI-Modell gebildet und an unternehmensindividuelle Anforderungen angepasst. So kann sich je nach Unternehmen eine andere Struktur ergeben [Dur08]. Als Grundlage für eine unternehmensspezifische Strukturierung kann das *IT Infrastructure Framework* nach Liu herangezogen werden [Liu02], welches vordefinierte Layer wie beispielsweise die *Anwendungsschicht* (business applications), die *Entwicklungsschicht* (development environment), die *Rechnerschicht* (computing environment) und die *Netzwerkschicht* (networking environment) beinhaltet. Neben diesen Layern bietet das Framework auch entsprechende Cluster zur funktionalen Gruppierung von Komponenten. Im Kontext dieser Arbeit bezeichnen wir Cluster als *Kategorien* und spezifizieren diese mit Hilfe der folgenden Definition.

Definition 2.16: Kategorie

Ein *Kategorie* bündelt technologische Komponenten, die vom gleichen Typ im Hinblick auf ihre Funktionalität sind, innerhalb einer Gruppe. Zudem ist eine Kategorie funktional einem spezifischen Layer zugeordnet.

Kategorien können demnach verwendet werden, um eine weitere Konkretisierung der funktionalen Einordnung von technologischen Komponenten zu ermöglichen. Im Rahmen des *IT Infrastructure Framework* werden hierfür beispielsweise die Kategorien *Datenbankserver*, *Applikationsserver* und *Integrationsserver* im Layer Entwicklungsschicht vorgeschlagen [Liu02]. So würde beispielsweise die technologische Komponente `MySQL 5.6` der Kategorie *Datenbankserver* zugeordnet werden. Solch eine Struktur lässt sich mit Hilfe eines *Kategoriensystems* weiter präzisieren.

Definition 2.17: Kategoriensystem

Als *Kategoriensystem* kann eine mehrstufige Ordnung verstanden werden, welche eine zusammenhängende Menge von hierarchischen Verbindungen zwischen Ober- und Unterkategorien abbildet.

Ein beispielhaftes Kategoriensystem für Datenbanksysteme ist in Abbildung 2.8 dargestellt. Wie hier zu erkennen ist, führt die Oberkategorie *Datenbanksystem (DBS)* zu den beiden Unterkategorien *Relationales Datenbanksystem (RDBS)* und *Hierarchisches Datenbanksystem (HDBS)*, wobei die Kategorie *RDBS* weiter in *Proprietäres* und *Open Source RDBS* unterteilt ist. Die technologische Komponente `MySQL 5.6` ließe sich demnach in die präzisere Kategorie *Open Source RDBS* einordnen.

Um auch die Struktur von Softwaresystemen mit einer Multi-Tier-Architektur, also zum Beispiel typische Client-Server Anwendungen, im Rahmen der Technologiearchitektur berücksichtigen zu können, muss das beschriebene Schichtenmodell um sogenannte *Tiers* erweitert werden.

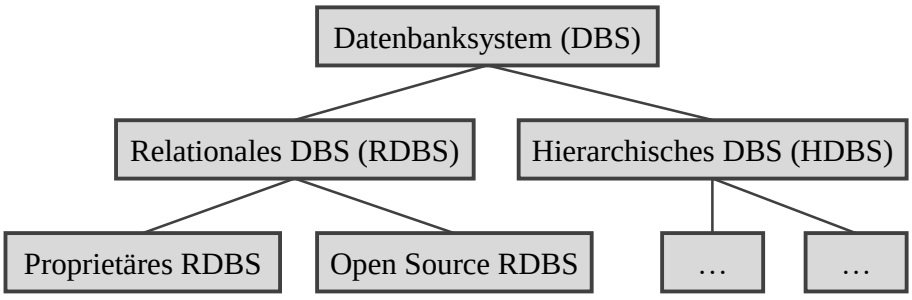


Abbildung 2.8: Beispielhaftes Kategoriensystem für Datenbanksysteme

Definition 2.18: Tier

Ein *Tier* ist eine vertikale Schicht innerhalb einer Technologiearchitektur und ermöglicht die Abbildung von technologischen Komponenten auf physikalisch getrennten Teilen des betrachteten Softwaresystems.

Typischerweise stellen Tiers physikalisch separierte Systembestandteile, wie beispielsweise dedizierte *Clients*, *Anwendungsserver* oder *Datenbankserver* dar. Diese lassen sich im Kontext einer systemspezifischen Technologiearchitektur betrachten und können auch an unternehmensindividuelle Anforderungen und Rahmenbedingungen angepasst werden.

Auf Grundlage der entsprechenden Definitionen für Tiers, Kategorien und das Kategoriensystem erweitern wir das vorgestellte Schichtenmodell für den Kontext der vorliegenden Arbeit und konkretisieren die Komponentensicht als *Technologiearchitekturplan (TAP)* mit Hilfe der folgenden Definition.

Definition 2.19: Technologiearchitekturplan

Als *Technologiearchitekturplan (TAP)* bezeichnen wir die logische und funktionale Struktur der Technologiearchitektur eines einzelnen Softwaresystems, welche durch einen Ordnungsrahmen basierend auf Layern, Tiers und Kategorien (oder alternativ einem Kategoriensystem) definiert ist.

Dabei leitet sich der Begriff Technologiearchitekturplan von dem Terminus *IT-Bebauungsplan* aus dem Gebiet der Softwarekartographie ab, welcher häufig eingesetzt wird, um logische Strukturen in Form einer Clusterkarte zu beschreiben [Mat08]. In Abbildung 2.9 ist die Struktur eines TAPs schematisch dargestellt. Wie hier zu erkennen ist, kann ein entsprechender TAP über eine beliebige Menge von Layern, Tiers und Kategorien verfügen. Somit kann eine konkrete TAP-Struktur individuell erstellt und an spezifische Unternehmensanforderungen angepasst werden.

Für die spätere Analyse von TAPs definieren wir zusätzlich die *Intersection* als weiteres Element dieses logisch funktionalen Ordnungsrahmens.

Definition 2.20: Intersection

Eine *Intersection* ist ein Element der logisch funktionalen Struktur eines Technologiearchitekturplans und stellt einen Schnittpunkt zwischen einem konkreten Layer und einem bestimmten Tier dar.

So kann ein entsprechender Schnittpunkt in der Form *Layer-Tier-Intersection* definiert werden. Demnach lässt sich ein beispielhafter Schnittpunkt zwischen

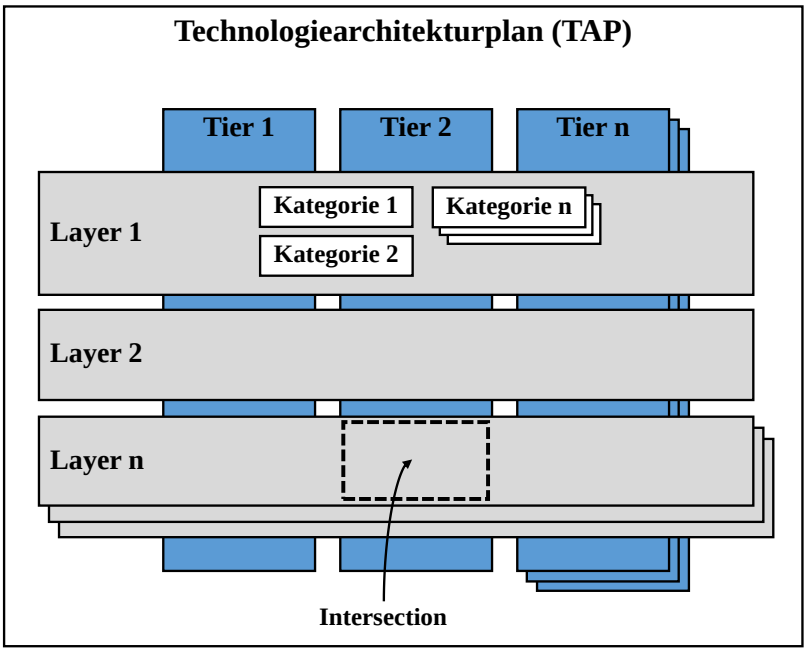


Abbildung 2.9: Strukturierung eines Technologiearchitekturplans (TAP)

einem Layer *Rechnerschicht* und einem Tier *Datenbankserver* als *Rechnerschicht-Datenbankserver-Intersection* beschreiben.

In der folgenden Tabelle 2.1 ist ein konkreter TAP für ein beispielhaftes Client-Server-Anwendungssystem dargestellt. Die logisch funktionale Struktur dieses TAPs ist durch die typischen Layer *Hardware*, *Betriebssystem*, *Daten*, *Applikation* und *Präsentation* sowie durch die Tiers *Client* und *Server* definiert.

Dieser exemplarische TAP beinhaltet alle technologischen Komponenten, sowohl Hardware- als auch Softwarekomponenten, die im Rahmen der spezifischen Technologiearchitektur im betrachteten Client-Server-Anwendungssystem verbaut sind. So be-

	Client	Server
Präsentation	–	phpMyAdmin 4.6
Applikation	Firefox 42 –	Tomcat 7 OpenLDAP 2.1
Daten	–	MySQL 5.6
Betriebssystem	Windows 7	Redhat 6
Hardware	x86	x86-Server

Tabelle 2.1: Konkreter TAP für ein beispielhaftes Client-Server-System

findet sich beispielsweise die Komponente `x86` auf der *Hardware-Client-Intersection*, die Komponente `Redhat 6` auf der *Betriebssystem-Server-Intersection* und die Komponente `Tomcat 7` auf der *Applikation-Server-Intersection*. Aus Gründen der Übersichtlichkeit sind die entsprechenden Kategorien hier nicht mit aufgeführt, sind aber als Konstrukt der Gesamtstruktur Bestandteil dieses TAPs. So ist beispielsweise die Komponente `MySQL 5.6` der Kategorie *RDBS* und die Komponente `Firefox 42` der Kategorie *Browser* zugeordnet.

2.3 Domänenspezifische Sprache

Im Rahmen der Softwareentwicklung bietet eine *Domänenspezifische Sprache*, im englischen als *Domain-Specific Language (DSL)* bezeichnet, die Möglichkeit, sich auf relevante Aspekte einer spezifischen Problemdomäne zu fokussieren. So stellt eine DSL konkrete Notationen für solch eine Domäne zur Verfügung, um komplexe Sachverhalte und wiederkehrende Probleme mit wenig Aufwand zu beschreiben [VBD+13].

Definition 2.21: Domain-Specific Language

Eine Domain-Specific Language (DSL) ist eine kleine, im Normalfall deklarative Sprache, welche Notationen und Abstraktionen für eine bestimmte Problemdomäne anbietet. Solche DSLs werden auch als *Micro-Languages* oder *Little Languages* bezeichnet und stellen oft eine ausführbare Spezifikationssprache dar [vKV00].

Bei näherer Betrachtung lassen sich DSLs nach der Art und Weise ihrer Realisierung in die drei Kategorien *intern*, *extern* und *Language Workbench-basiert* unterteilen [FP10]. Während eine interne DSL[2] in eine allgemein nutzbare General-Purpose Language (GPL) eingebettet ist und damit auf die Konzepte der Hostsprache (Gastgebersprache) aufsetzt, zeichnen sich externe DSLs[3] durch eine unabhängige, frei definierbare und auf die betrachtete Domäne fokussierte Syntax aus [FP10, CV09]. Eine Language Workbench[4] bietet für die Entwicklung solcher DSLs eine spezialisierte, Tool-basierte Umgebung, mit deren Hilfe Experten frei definierte Sprachen erstellen und miteinander integrieren können [FP10]. Die Vorteile einer eigenständigen DSL liegen dabei beispielsweise in der effizienten und produktiven Nutzung solch einer Sprache für domänenspezifische Anwendungsfälle sowie in der guten Handhabbarkeit für Domänenexperten [VBD+13, DS15].

Zur Definition einer DSL müssen ihre *abstrakte* und *konkrete Syntax* sowie ihre *Semantik* und eventuelle *Kontextbedingungen* beschrieben werden [HR04]. Während

[2] Beispiel für eine interne DSL: eXtensible Application Markup Language (XAML) auf Basis der Microsoft Plattform .Net [CV09]
[3] Beispiel für eine externe DSL: Hypertext Markup Language (HTML) [VBD+13]
[4] Beispiel für eine Language Workbench: MontiCore [GKR+08, KRV08b]

die abstrakte Syntax die Elemente einer Sprache und deren Beziehungen festlegt und oftmals mit Hilfe von Metamodellen definiert wird, repräsentiert die konkrete Syntax die Menge aller zulässigen Sprachinstanzen, welche von Nutzern der DSL verwendet werden können [KRV08a, Kra10]. Solch eine konkrete Syntax wird oftmals durch eine textuelle Notation (Grammatik), welche eine DSL in Sätze, Wörter und Zeichen zerlegt [CV09], oder durch eine grafische Spezifikation beschrieben [KRV08a]. Die Semantik definiert die Bedeutung der entsprechenden Syntax einer DSL und Kontextbedingungen können Einschränkungen für die Verwendung der jeweiligen Sprachinstanzen definieren, um die Wohldefiniertheit der Sprache sicherzustellen [KRV08a, Kra10].

Im Rahmen dieser Arbeit werden DSLs verwendet, um Anforderungen von Domänenarchitekten mit wenig manuellem Aufwand zu spezifizieren, damit diese für die Analyse von TAPs und ihren Komponenten automatisiert verarbeitet werden können (vgl. Kapitel 4 und 5). Hierfür wurden entsprechende DSLs mit Hilfe von *Xtext*[5], einem Framework zur Erstellung von Programmiersprachen und DSLs, entwickelt. Dessen Notation ist an der syntaktischen Metasprache *Extended Backus-Naur Form (EBNF)* [ISO96] angelehnt, welche wesentliche Aspekte für die Entwicklung von Grammatiken für formale Sprachen definiert und dafür zwei wichtige Grundelemente einführt: *Terminale* und *Nicht-Terminale*. Terminale stellen eine Sequenz von einem oder mehreren Zeichen dar und sind ein nicht reduzierbares Element einer Sprache, zum Beispiel `INT='0'..'9'` als numerisches Terminal für natürliche Zahlen von 0 bis 9. Dagegen entsprechen Nicht-Terminale einem syntaktischen Teil einer Sprache, der weiter definiert werden muss [ISO96], beispielsweise `ID` als Identifikationsnummer.

In der folgenden Tabelle 2.2 sind die für diese Arbeit relevanten Xtext-Notationen im Vergleich zu den entsprechenden EBNF-Notationen dargestellt. Die hier aufgeführten Xtext-Notationen können für die Definition der konkreten Syntax einer DSLs verwendet werden. So beschreibt beispielsweise `enum Farbe: (ROT = 'Rot'| GRUEN = 'Grün'| BLAU = 'Blau');` die Definition (`:`) und Terminierung (`;`) einer Syntaxregel für das Nicht-Terminal `Farbe`, welches eine Auswahl (`enum`) der Alternativen (|) `Rot, Grün` und `Blau` zur Beschreibung einer Farbe anbietet. Dabei stellen die jeweiligen Alternativen String-Terminale (z.B. `'Rot'`) dar, welche einem DSL-Feature[6] zugewiesen (`=`) sind (z.B. `ROT`) und nicht weiter definiert werden müssen. Zudem lässt sich durch das Hinzufügen der Operatoren `?,*,+` an das Ende der Alternativgruppe definieren, wie oft eine Farbe auswählbar ist. Solch eine Farbauswahl kann dann einem weiteren DSL-Feature, zum Beispiel (`AUTOFARBE`), durch eine zusätzliche Grammatikregel `FahrzeugAussen: AUTOFARBE = Farbe;` zugewiesen (`=`) werden. Über den Operator `+=` ließe sich auch eine mehrfache Farbauswahl zuweisen, um beispielsweise

[5] https://www.eclipse.org/Xtext

[6] Ein DSL-Feature beschreibt ein Objekt der abstrakten Syntax (Metamodell) einer DSL [Yue14]

	EBNF-Notation	**Xtext-Notation**
Definition	=	:
Verkettung	,	(direktes Verketten ohne Notation)
Terminierung	;	;
Alternation	|	|
Option (0 oder 1)	[...]	(...)?
Wiederholung (0 bis ∞)	...	(...)*
Wiederholung (1 bis ∞)	(keine Notation)	(...)+
Gruppierung	(...)	(...)
Terminal String	"..."	'...'
Bereich	(keine Notation)	..
Zuweisung	(keine Notation)	=
Mehrfachzuweisung	(keine Notation)	+=

Tabelle 2.2: EBNF- und Xtext-Notationen im Vergleich nach [Yue14]

einen Farbverlauf zu beschreiben. Alternativ zu den String-Terminalen können die Farben auch als nummerische Werte spezifiziert werden, indem die jeweiligen Terminale in der Form `Terminal ROT: ('0'..'9')+;` definiert werden. Dies beschreibt eine Auswahl von mindestens einer natürlichen Zahl im Bereich `0` bis `9`. Durch Auswahl von drei auf einander folgenden Ziffern kann so ein entsprechender Farbton im RGB-Farbraum abgebildet werden.

3 Analyse der Variabilität von Technologiearchitekturen

Dieses Kapitel basiert in wesentlichen Teilen auf den Veröffentlichungen [WWS+17], [WWSS17b] und setzt dabei auf erste Ideen in [WWPS16] auf.

Um die Variabilität von Technologiearchitekturen bestimmen zu können, müssen die TAPs von betrachteten Softwaresystemen analysiert werden, um so die konkreten Variabilitätsbeziehungen zwischen den verbauten Komponenten zu identifizieren. Da diese Aufgabe bisher von Domänenexperten manuell durchgeführt werden muss, ist die Identifizierung der Variabilität von Technologiearchitekturen sehr zeitaufwendig. Zudem ist es kaum machbar, solch eine manuelle Analyse für hunderte Softwaresysteme in einer gewachsenen IT-Landschaft mit tausenden von unterschiedlichen technologischen Komponenten durchzuführen.

Zur Lösung dieser Problematik wird in diesem Kapitel ein modellbasiertes Mining-Verfahren vorgestellt, welches in der Lage ist, eine beliebige Menge von TAPs gleichzeitig zu analysieren und detaillierte Variabilitätsbeziehungen zwischen den implementierten technologischen Komponenten automatisiert zu bestimmen.

Die folgende Abbildung 3.1 zeigt eine schematische Darstellung des Workflows für unseren Ansatz zur Analyse der Variabilität in Technologiearchitekturen. Wie hier zu sehen ist, werden zuerst verschiedene Maßnahmen durchgeführt, die erforderlich sind, um die verfügbaren Daten über TAPs für die darauf folgende Variabilitätsbestimmung vorzubereiten (vgl. Abschnitt 3.1). Dies beinhaltet im Wesentlichen die

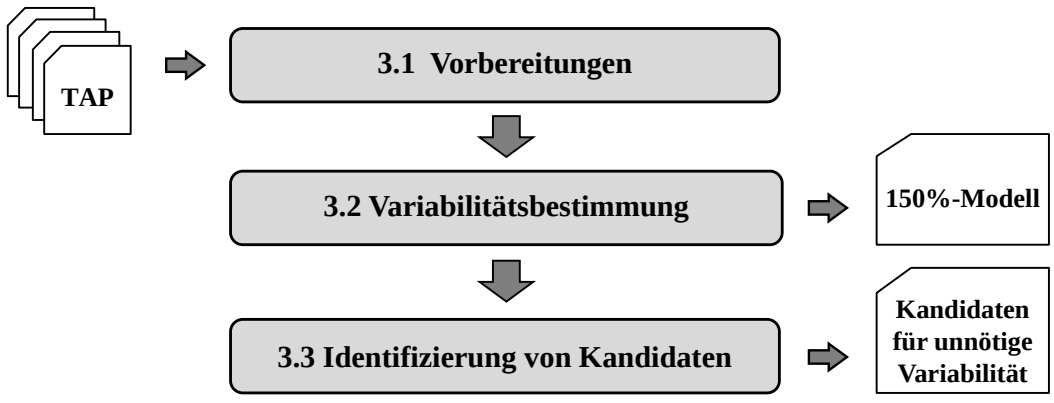


Abbildung 3.1: Workflow für die Analyse von Variabilität

beiden *Phasen A* und *B* unseres Gesamtansatzes (vgl. Abbildung 1.1). Danach erfolgt die Durchführung unseres Mining-Verfahren zur Bestimmung der Variabilität in den betrachteten Technologiearchitekturen (vgl. Abschnitt 3.2). Dies umfasst die *Phase C* unseres Gesamtansatzes. Im letzten Schritt werden dann auf Basis der Analyseergebnisse diejenigen Komponenten identifiziert, die als Kandidaten für unnötige Variabilität betrachtet werden können (vgl. Abschnitt 3.3).

3.1 Vorbereitung

Bevor unser Variabilitätsmining-Verfahren die TAPs verschiedener Softwaresysteme analysieren kann, sind einige Vorbereitungen erforderlich. Wie in Abbildung 3.2 zu erkennen ist, sind dazu im ersten Schritt *Datenauswahl* (vgl. Unterabschnitt 3.1.1) geeignete TAPs für die spätere Analyse zu identifizieren und zu einem Set zusammenzuführen. Anschließend müssen diese TAPs im zweiten Schritt *Datentransformation* (vgl. Unterabschnitt 3.1.2) importiert und in das interne Datenmodell überführt werden, wodurch von unnötigen Details abstrahiert und eine automatisierte Verarbeitung ermöglicht wird. Im letzten Schritt *Datenaufbereitung* (vgl. Unterabschnitt 3.1.3) wird dann die Qualität der importierten und transformierten Daten von TAPs analysiert und bei Bedarf durch geeignete Maßnahmen verbessert, um so präzisere Ergebnisse im Rahmen der Variabilitätsbestimmung zu ermöglichen.

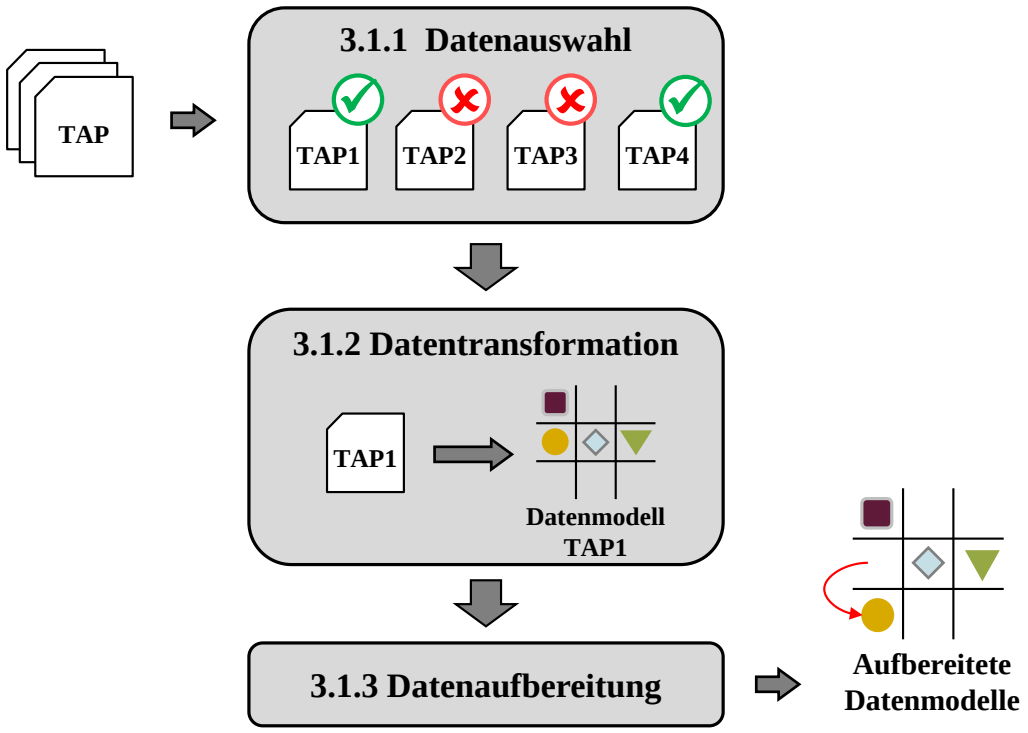


Abbildung 3.2: Workflow für die Vorbereitung

3.1.1 Datenauswahl

Unser Variabilitätsmining-Verfahren identifiziert alle Gemeinsamkeiten und Unterschiede zwischen betrachteten TAPs. Um im Rahmen dieses Verfahrens sinnvolle Variabilitätsbeziehungen identifizieren zu können, ist es erforderlich, dass Domänenexperten geeignete TAPs für die Analyse auswählen. Eine Menge von TAPs wird dann als geeignet betrachtet, wenn alle beinhalteten TAPs den gleichen allgemeinen Verwendungszweck erfüllen.

Definition 3.1: Allgemeiner Verwendungszweck von TAPs

Der *allgemeine Verwendungszweck* eines TAPs ist durch dessen technologisches Hauptmerkmal beschrieben. Dieses kann entweder durch die implementierte Systemarchitektur oder durch die eingesetzte Kerntechnologie bestimmt sein. Auch eine Kombination beider Merkmale kann zu einem allgemeinen Verwendungszweck zusammengefasst werden.

So können beispielsweise TAPs von Softwaresystemen ausgewählt werden, welche alle die Systemarchitektur für eine *Web-Client-Applikation* oder für eine *FAT-Client-Applikation* implementieren. Darüber hinaus können auch TAPs ausgewählt werden, welche als Kerntechnologie beispielsweise *SAP*, *.Net* oder *Java* einsetzen. Zudem können Domänenexperten auch solche TAPs für die Analyse auswählen, die eine Kombination der beiden Merkmale erfüllen (z.B. *Java-Web-Client-Applikationen*).

Erst durch die Dekomposition einer TA in Sets von TAPs mit gleichem Verwendungszweck ist es möglich, sinnvolle Variabilitätsinformationen abzuleiten und für identifizierte Technologievarianten konkrete Restrukturierungsentscheidungen zu treffen. Andernfalls würde die Analyse von TAPs unterschiedlichster Applikationen lediglich zur Identifizierung von Unterschieden führen, da gewachsene IT-Landschaften häufig hunderte von verschiedenen Softwaresystemen beinhalten, welche in einer Gesamtbetrachtung kaum Gemeinsamkeiten aufweisen würden. Zur Auswahl geeigneter TAPs für die Variabilitätsanalyse ist demnach zuerst der gemeinsame Verwendungszweck zu bestimmen, bevor die Domänenexperten in einem zweiten Schritt die entsprechenden TAPs für die Analyse selektieren und zu einem Set zusammenfassen können.

In der nachfolgenden Tabelle 3.1 wird ein fortlaufendes Beispiel für diese Arbeit vorgestellt. Es zeigt ein Set von vier TAPs unterschiedlicher Anwendungssysteme, die für die spätere Variabilitätsanalyse ausgewählt wurden. Jeder dieser TAPs weist die fünf Layer *Hardware*, *Betriebssystem*, *Daten*, *Applikation* und *Präsentation* sowie die beiden Tiers *Client* und *Server* auf. Zudem beinhalten sie die technologischen Komponenten, die für den Betrieb der jeweiligen Applikationen benötigt werden. Für die Auswahl dieser TAPs ist zuerst der allgemeine Verwendungszweck *Java-Web-Client-Applikation* festgelegt worden. Anschließend wurden diese TAPs anhand ihrer eingesetzten Komponenten identifiziert. Wie zu erkennen ist, beinhaltet jeder dieser vier

	Client	Server
Präsentation	**Firefox 42**	phpMyAdmin 4.6
Applikation	–	**Tomcat 7**
	–	OpenLDAP 2.1
Daten	–	MySQL 5.6
Betriebssystem	Windows 7	Redhat 6
Hardware	x86	x86-Server

(a) TAP1

	Client	Server
Präsentation	**Firefox 42**	–
Applikation	–	**Tomcat 6**
Daten	–	–
Betriebssystem	Windows 10	CentOS 6
Hardware	x86	x86-Server

(b) TAP2

	Client	Server
Präsentation	**Firefox 49**	–
Applikation	–	**WebSphere App. Server 7.0**
	–	Tivoli Access Manager 6.1
Daten	–	IBM DB2 10
Betriebssystem	Windows 7	Redhat 6
Hardware	x86	x86-Server

(c) TAP3

	Client	Server
Präsentation	**Internet Explorer 8**	phpMyAdmin 2.5
	Firefox 42	–
Applikation	–	**Tomcat 7**
Daten	–	MySQL 5.7
Betriebssystem	Windows 7	Redhat 7
Hardware	x86	x86-Server

(d) TAP4

Tabelle 3.1: Vier TAPs von verschiedenen IT-Systemen als fortlaufendes Beispiel

TAPs einen Java-Webserver (`Tomcat` oder `WebSphere Application Server`) sowie einen entsprechenden Browser (`Firefox` oder `Internet Explorer`) für die Benutzer-Interaktion.

3.1.2 Datentransformation

Unser Variabilitätsmining-Ansatz erwartet ein Set von beliebig vielen TAPs, welche detaillierte Strukturinformationen über die TAs der zu analysierenden Softwaresysteme liefern. Um solche TAPs automatisiert verarbeiten zu können, müssen sie allerdings in einem spezifischen Format vorliegen. Hierzu wird in der Vorphase *Datentransformation* jeder vorliegende TAP in ein internes Datenmodell überführt. Damit solche Datenmodelle von unserem Mining-Ansatz verarbeitet werden können, definieren wir ein Metamodell, welches das spezifische Format für konkrete Datenmodelle beschreibt.

Definition 3.2: Metamodell

Ein Metamodell ist ein Modell eines Modells, welches die Sprache des untergeordneten Modells mit den zu verwendenden Sprachartefakten beschreibt [Str98].

Mit Hilfe eines Metamodells können benötigte Daten für die nachfolgende Clusteranalyse berücksichtigt und Informationen, die für unser Variabilitätsmining-Ansatz nicht erforderlich sind, abstrahiert werden. Beispielsweise sind TAPs häufig mit zusätzlichen Daten, wie dem Namen und der E-Mail-Adresse von verantwortlichen Personen im Unternehmen angereichert. Durch die Abstraktion solcher nicht erforderlichen Daten, erlauben die konkreten Datenmodelle als Instanz unseres Metamodells eine Fokussierung auf die wesentlichen Informationen für die Variabilitätsanalyse, ohne dabei unnötige Details zu verarbeiten. Zudem liefert das Metamodell auch eine klare Übersicht darüber, welche Daten benötigt werden und wie sie für unseren Mining-Ansatz strukturiert sein müssen. So ist es möglich, erforderliche Informationen über TAPs aus verschiedenen Datenquellen (z.B. aus Datenbanken oder CSV-Tabellen) zu verwenden.

In Abbildung 3.3 stellen wir das Metamodell für unseren Ansatz vor. Jeder TAP besteht aus `TAP-Komponenten`, die einem konkreten `Physischen Element` entsprechen, welches auf einem bestimmten `Layer` und `Tier` im TAP implementiert ist (z.B. `Windows Server 2013` auf dem `Betriebssystem-Layer` und dem `Server-Tier`). Durch die Modellierung von TAP-Komponenten ist es möglich, mehrere Instanzen desselben physischen Elementes auf verschiedenen Tiers zu beschreiben. So kann der `Windows Server 2013` beispielsweise sowohl auf dem `Server-Tier` als auch auf einem dedizierten `Datenbank-Tier` verbaut sein. Um solche physischen Elemente zu gruppieren, verwenden wir das `Logische Element`. Es beschreibt ein logisches Objekt, welches Verweise auf alle physische Element beinhal-

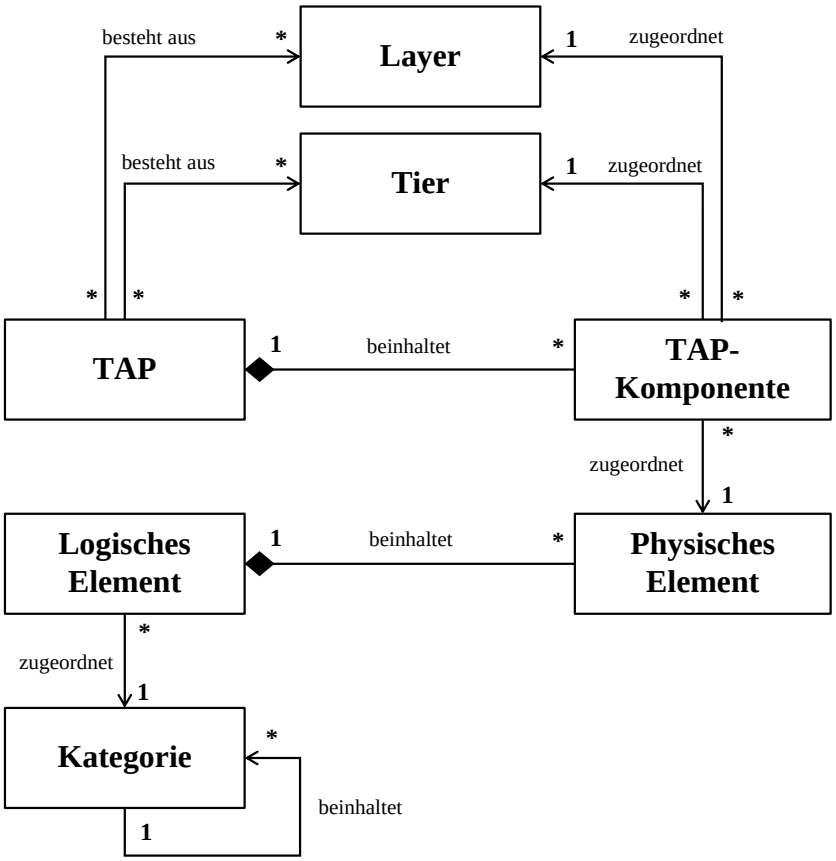


Abbildung 3.3: Metamodell für die Repräsentation von TAPs

tet, die eine konkrete, implementierbare Version des logischen Elementes darstellen. So sind beispielsweise die beiden physischen Elemente `Windows Server 2013` und `Windows Server 2016` einsetzbare Komponenten des gleichen logischen Elementes `Windows Server`. Darüber hinaus können logische Elemente je nach der von ihnen bereitgestellten Funktionalität in verschiedene `Kategorien` eingeteilt werden. Zum Beispiel entspricht das logische Element `Windows Server` der Kategorie *Betriebssystem*, während das logische Element `MySQL` der Kategorie *Datenbank* zugeordnet ist.

Abhängig von der Anzahl der verschiedenen Komponenten in diesen Kategorien kann es nützlich sein, ein *hierarchisches Kategoriensystem* aufzubauen, indem die Kategorien der obersten Ebene in Unterkategorien unterteilt werden. Dies ermöglicht später eine feingranularere Variabilitätsanalyse. Hierzu wird dem logischen Element eine detailliertere Unterkategorie zugeordnet, welche in einer Oberkategorie enthalten ist. So könnte beispielsweise die Oberkategorie *Betriebssystem* in die Unterkategorien *Windows-basiertes-Betriebssystem* und *Unix-basiertes-Betriebssystem* unterteilt werden, um eine bessere Unterscheidung von Windows- und Linux-Serversystemen zu ermöglichen.

In seltenen Fällen kann eine Komponente auch mehreren Kategorien zugeordnet werden. Zum Beispiel könnte der *Webbrowser* `Internet Explorer` auch als *User-Interface* einer Anwendungsplattform zum Ausführen von Anwendungen angesehen werden (z.B. für das *HP Quality Center*). Unser Ansatz beruht jedoch auf der eindeutigen Zuordnung von Komponenten zu einer einzelnen Kategorie, um die Komplexität zu reduzieren. Daher sollten Komponenten eindeutig der Kategorie zugeordnet werden, die dem Hauptzweck der jeweiligen Komponente entspricht.

Design-Entscheidung 3.1: Umfang des Metamodells

Umfangreiche Informationen über TAPs sind möglicherweise nicht in jedem Unternehmen verfügbar. Um unseren automatisierten Ansatz dennoch für eine Vielzahl von Anwendern nutzbar zu machen, haben wir uns auf die im Metamodell gezeigten Daten beschränkt. Hierdurch ist es möglich, bereits mit Basis-Informationen die Variabilität von betrachteten TAPs zu bestimmen. Durch eine Erweiterung des Metamodells können auch zusätzliche Informationen (z.B. Funktionen oder Kosten) berücksichtigt werden. Solche detaillierteren Informationen über Eigenschaften von Komponenten können dabei helfen, eine noch feingranularere Variabilitätsanalyse durchzuführen.

In Abbildung 3.4 ist die Datentransformation schematisch dargestellt. Reale TAPs aus verschiedenen Datenquellen (z.B. Datenbanken oder CSV-Dateien) werden durch geeignete *Importer* angebunden und in konkrete TAP-Datenmodelle des Zielformates überführt. Dieses Zielformat wird dabei durch das verwendete TAP-Metamodell (vgl. Abbildung 3.3) beschrieben. Für jeden einzelnen TAP wird so eine entsprechende Repräsentation in Form eines konkreten TAP-Modells erzeugt.

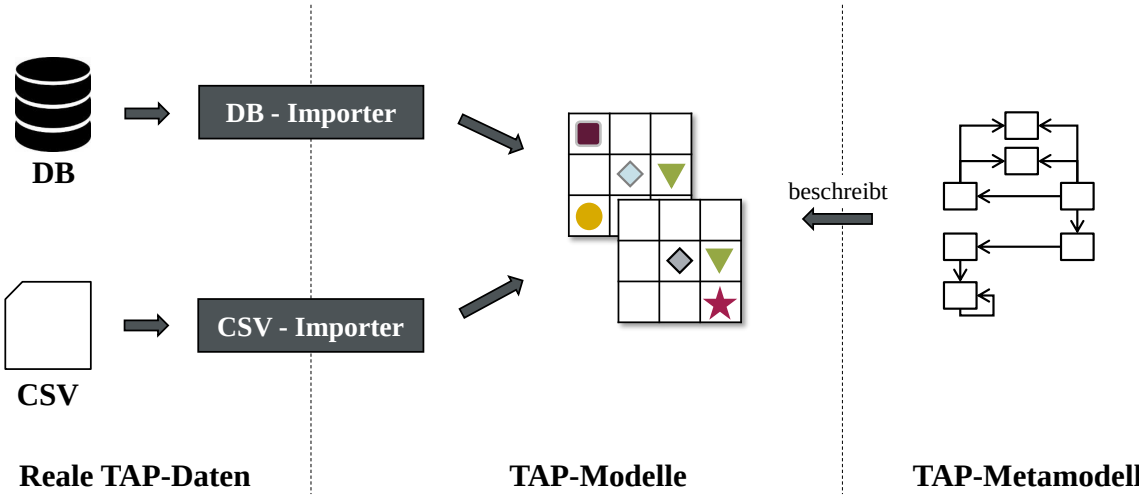


Abbildung 3.4: Schematische Darstellung der Datentransformation

3.1.3 Datenaufbereitung

Unser Variabilitätsmining-Verfahren ist, wie potentiell jedes automatisierte Analyse-Verfahren, in hohem Maße von der Qualität der bereitgestellten Daten abhängig. Während der Analyse der verfügbaren Daten unseres Industriepartners konnten wir unterschiedliche Level der Datenqualität feststellen. Insgesamt waren die verfügbaren Daten über TAPs gut gepflegt und enthielten wenige Fehler. Da die Dokumentation dieser Daten jedoch vollständig manuell erfolgte, haben wir kleinere Fehler entdeckt, die versehentlich eingeführt wurden. Oftmals handelte es sich dabei um falsche Zuordnungen von Komponenten zu Layern oder Tiers. Auch ohne spezifisches Domänenwissen wurden diese Fehler offensichtlich, da beispielsweise Komponenten auf Ebenen abgebildet wurden, die keinen Sinn ergaben (z.B. haben wir die Betriebssystem-Komponente `Redhat 6` auf dem obersten Layer *Präsentation* identifiziert).

Da solche Ungenauigkeiten zu fehlerhaften Ergebnissen in der Variabilitätsanalyse führen können, sollten sie entsprechend berücksichtigt werden. Zu diesem Zweck beinhaltet unser Ansatz die beiden Methoden *Strukturanalyse* und *Ausreißersuche*, mit deren Hilfe Mängel an der Datenqualität der zu analysierenden TAPs erkannt und automatisiert oder semi-automatisiert beseitigt werden können. So lassen sich die Daten der zu analysierenden TAPs aufbereiten, bevor der eigentliche Prozess zur Variabilitätsbestimmung in der nächsten Phase (vgl. Abschnitt 3.2) erfolgt.

Strukturanalyse

Im Rahmen der Strukturanalyse werden alle Komponenten eines TAPs daraufhin untersucht, ob sie dem richtigen Layer zugeordnet sind. Da Layer eine vertikal ausgerichtete Möglichkeit zur funktionalen Klassifikation von Komponenten darstellen (vgl. Unterabschnitt 2.2.2) und Komponenten selbst häufig einer funktionalen Kategorie zugewiesen sind, lässt sich durch die Kombination dieser Informationen ermitteln, ob sich eine Komponente auf dem richtigen Layer befindet. Hierfür wird die Struktur des verwendeten Kategoriensystems herangezogen und, falls noch erforderlich, um die entsprechenden Layer der betrachteten TAPs erweitert. Dabei ist eine Komponentenkategorie immer eindeutig einem Layer zuzuordnen, da sich solche Kategorien in die funktionale Struktur der Layer einordnen müssen. So erfolgt eine Zuordnung von Layern zu Komponentenkategorien, welche anschließend als Strukturdefinition zur Verfügung steht. In der folgenden Tabelle 3.2 ist ein Beispiel für solch eine Strukturdefinition aufgeführt. Diese weist den beiden Layern *Präsentation* und *Betriebssystem* jeweils eine Oberkategorie mit drei Unterkategorien zu.

Im Rahmen der Strukturanalyse werden die Datenmodelle der jeweiligen TAPs im Hinblick auf die vorliegende Strukturdefinition analysiert. Dazu wird für alle in einem Modell verbauten Komponenten ihr jeweiliges logisches Element bestimmt und die dazu gehörende Kategorie identifiziert. Nun kann über ein Vergleich der ermittelten Kategorie mit der spezifizierten Kategorie aus der Strukturdefinition ein Mapping der entsprechenden Layer erfolgen. Stimmt dabei der vordefinierte Layer der Struk-

Layer	Oberkategorie	Unterkategorie
Präsentation	Viewer	Technology Specific Viewer
		Multimedia Viewer
		Internet Browser
Betriebssystem	Client Operating Systems	Mobile Devices OS
		Desktop OS
		Workstation OS

Tabelle 3.2: Beispielhafte Strukturdefinition

turdefinition nicht mit dem tatsächlich im TAP zugeordneten Layer überein, so liegt ein Zuordnungsfehler vor. Dieser Fehler kann durch ein automatisiertes Ersetzen des Wertes für den Layer im entsprechenden Datenmodell durch den korrekten Layer aus der Strukturdefinition bereinigt werden.

In Tabelle 3.3 ist solch eine falsche Layer-Zuordnung beispielhaft anhand der Komponente `Firefox 42` im `TAP1` des fortlaufenden Beispiels (vgl. Tabelle 3.1) dargestellt. Hierbei ist der `Firefox 42` (Unterkategorie *Internet Browser*) dem Layer *Applikation* zugeordnet worden, wobei der Layer *Präsentation* als korrekt aus der Strukturdefinition (vgl. Tabelle 3.2) abgeleitet werden kann. Diese Fehlzuordnung lässt sich nun automatisiert bereinigen, indem der Wert `Applikation` des entsprechenden Layers für die Komponente `Firefox 42` im Datenmodell des `TAP1` mit dem richtigen Wert `Präsentation` aus der Strukturdefinition ersetzt wird.

Durch die Anwendung der Strukturanalyse können Datenfehler bereinigt werden, die auf eine falsche Zuordnung von Komponenten zu Layern zurückzuführen sind. Diese Technik lässt sich allerdings nicht für die Identifikation von fehlerhaften Zuordnungen zu Tiers nutzen, da Komponenten durchaus auf unterschiedlichen Tiers eingesetzt werden können. Um aber auch die Erkennung und Bereinigung solcher Fehlzuordnungen zu ermöglichen, nutzen wir die nachfolgende Ausreißersuche.

	Client	Server
Präsentation	–	phpMyAdmin 4.6
Applikation	**Firefox 42**	Tomcat 7
	–	OpenLDAP 2.1
Daten	–	MySQL 5.6
Betriebssystem	Windows 7	Redhat 6
Hardware	x86	x86-Server

Tabelle 3.3: TAP1 aus dem fortlaufenden Beispiel mit falsch zugeordnetem Browser

Ausreißersuche

Im Rahmen der Ausreißersuche werden Komponenten in betrachteten TAPs identifiziert, die potentiell dem falschen Tier zugeordnet sind. Da allerdings die Verwendung einer konkreten Komponente auf unterschiedlichen Tiers zulässig ist, kann eine falsche Zuordnung der entsprechenden Komponente nicht eindeutig identifiziert werden. Beispielsweise lässt sich die Betriebssystem-Komponente `Windows Server 2013` sowohl auf dem `Server-Tier` als auch auf einem dedizierten `Datenbank-Tier` einsetzen. Um dennoch falsch zugeordnete Komponenten erkennen zu können, wird deren Häufigkeitsverteilung untersucht. Anhand der ermittelten Häufigkeiten können potentielle Ausreißer identifiziert und den Domänenexperten zur weiteren Überprüfung vorgelegt werden. Zur Bestimmung der jeweiligen Verwendungshäufigkeit von Komponenten in allen betrachteten TAPs wird die folgende Gleichung 3.1 angewendet:

$$H_{k_i(t_j)} = \frac{N_T \; mit \; k_i \; auf \; t_j}{N_T \; mit \; l_e(k_i)} \tag{3.1}$$

Mit Hilfe dieser Gleichung kann die *Häufigkeit H* einer spezifischen Komponente k_i (z.B. `Firefox 42`) auf einem konkreten Tier t_j (z.B. `Client-Tier`) ermittelt werden. Dies erfolgt, indem die Anzahl N_T der TAPs mit k_i auf t_j ins Verhältnis zur Anzahl N_T der TAPs gesetzt werden, die eine beliebige Komponente des gleichen logischen Elementes l_e von k_i (z.B. `Firefox 49`) verwenden. In der folgenden Tabelle 3.4 wird dies beispielhaft für die Häufigkeitsverteilung von Firefox-Komponenten innerhalb eines Sets von 10 TAPs gezeigt. Hierbei ist für den `Firefox 42` eine Häufigkeit von H=50% auf dem `Client-Tier` sowie für den `Firefox 49` eine Häufigkeit von H=40% auf dem `Client-Tier` und H=10% auf dem `Server-Tier` bestimmt worden.

Zur Identifizierung von Ausreißern werden die ermittelten Häufigkeitswerte für jede Komponente k_i desselben logischen Elementes l_e auf dem gleichen Tier t_j zusammengefasst. Dies erfolgt, da unterschiedliche Versionen einer Komponente bei der Ausreißersuche nicht relevant sind. Als Ausreißer werden dann solche Komponenten betrachtet, deren Häufigkeit unter 15% und damit unter dem Schwellwert $S_H = 0,15$ für einen spezifischen Tier t_j liegt (vgl. Design-Entscheidung 3.2). In dem gezeigten Beispiel in Tabelle 3.4 ergibt sich somit für den `Firefox` eine Häufigkeit von H=90% für den `Client-Tier` und von H=10% für den `Server-Tier`. Demnach wird die Komponente `Firefox 49` auf dem `Server-Tier` als Ausreißer und damit als potentielle Falschzuordnung identifiziert.

Komponente k_i	**Log. Element l_e**	**Tier t_j**	**TAP**	**Häufigkeit $H_{k_i(t_j)}$**
Firefox 42	Firefox	Client	1,2,3,4,5	50%
Firefox 49	Firefox	Client	6,7,8,9	40%
Firefox 49	Firefox	Server	10	10%

Tabelle 3.4: Beispiel für die Ausreißersuche

Design-Entscheidung 3.2: Schwellwert S_H für die Ausreißersuche

Der Schwellwert S_H für die Identifizierung von Ausreißern ist hier mit 15% angegeben. Dieser Wert hat sich aus Gesprächen mit Experten unseres Industriepartners und der Analyse von verfügbaren Daten ergeben. Dabei stand im Vordergrund, dass eine geeignete Menge an TAPs für die Ausreißersuche vorliegen sollte, damit die Identifizierung von Ausreißern sinnvoll ist. Nichtsdestotrotz ist dieser Schwellwert frei konfigurierbar, sodass er an individuelle Rahmenbedingungen angepasst werden kann.

Da die identifizierten Komponenten nicht eindeutig als *falsch zugeordnet* klassifiziert werden können, ist zusätzliches Expertenwissen notwendig, um eine eindeutige Aussage bezüglich der Zuordnung zu den betrachteten Tiers treffen zu können. So liefert unser Ansatz den Experten die Häufigkeitswerte aller Komponenten für ihre jeweiligen Tiers und weist auf die identifizierten Falschzuordnungen hin. Zudem wird ein Vorschlag für eine *korrekte Zuordnung* gegeben, welcher den Tier mit der höchsten Häufigkeit für die betrachtete Komponente enthält. So können Domänenexperten eine identifizierte Falschzuordnung manuell verifizieren und bereinigen, wodurch der entsprechende Tier-Wert im Datenmodell des jeweiligen TAPs angepasst wird.

3.2 Verfahren zur Bestimmung der Variabilität in Technologiearchitekturen

Zur Bestimmung der Variabilität von Technologiearchitekturen haben wir ein Mining-Verfahren entwickelt, welches auf dem Variabilitätsmining-Ansatz nach Wille et al. [WSSS16, WRSS17, Wil18] aufsetzt und sich dabei die gut strukturierten Daten von TAPs zu Nutze macht. In Abbildung 3.5 ist der allgemeine Workflow für unser Variabilitätsmining-Verfahren schematisch dargestellt.

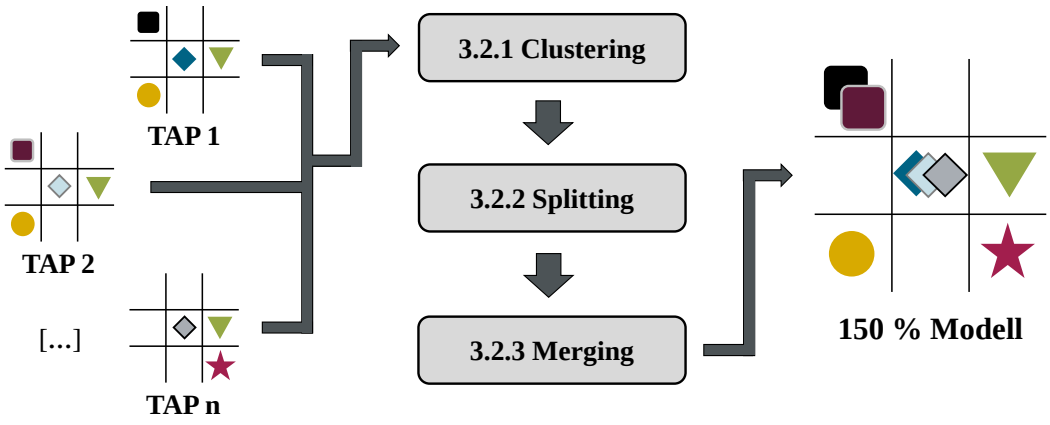


Abbildung 3.5: Workflow für das Variabilitätsmining-Verfahren

Wie in Abbildung 3.5 zu erkennen ist, erwartet unser Ansatz eine beliebige Menge von TAP-Modellen als Input. Darauf basierend werden in der *Clustering-Phase* (vgl. Unterabschnitt 3.2.1) die einzelnen, in den TAPs enthaltenen Komponenten unter Berücksichtigung ihrer strukturellen Beziehungen gruppiert, um so die Identifizierung unrealistischer Variabilitätsbeziehungen zwischen den einzelnen Komponenten zu vermeiden. Zur weiteren Verbesserung dieser Clustering-Ergebnisse, wird dann in der *Splitting-Phase* (vgl. Unterabschnitt 3.2.2) ein anpassbarer, regelbasierter Ansatz dazu verwendet, die Cluster weiter aufzusplitten, um potentielle Variabilitätsbeziehungen, die den Erwartungen von Domänenexperten widersprechen würden, zu entfernen. In der letzten *Merging-Phase* (vgl. Unterabschnitt 3.2.3) wird anschließend die Variabilität der identifizierten Komponenten-Cluster kategorisiert und in einem 150%-Modell zusammengefasst.

3.2.1 Clustering-Phase

In der Clustering-Phase werden geeignete *Cluster* von Komponenten gebildet, um potentielle Varianten von eingesetzten Technologiekomponenten über alle betrachteten TAPs hinweg identifizieren zu können.

Definition 3.3: Cluster

Cluster werden im Rahmen einer Clusteranalyse gebildet und stellen ein Klassifikationsschema zur Gruppierung von Individuen dar. Dabei beschreibt ein Cluster eine Klasse von Individuen, welche auf Basis ihrer individuellen Charakteristiken in einer gewissen Art und Weise gleich sind und sich dadurch von Individuen anderer Klassen unterscheiden [Eve11].

Nach der Transformation (vgl. Unterabschnitt 3.1.2) und Bereinigung (vgl. Unterabschnitt 3.1.3) der zu analysierenden Daten, sind wir nun in der Lage, die *Clusteranalyse* unseres Mining-Ansatzes für eine Menge von TAPs durchzuführen. Hiermit ist es möglich, Cluster von Komponenten zu bilden, die aufgrund ihrer Eigenschaften eine funktionale Ähnlichkeit zueinander aufweisen und damit potentielle Variabilitätsbeziehungen beinhalten. Durch die Gruppierung von Komponenten in funktionale Cluster stellen wir sicher, dass in der späteren Merging-Phase (vgl. Unterabschnitt 3.2.3) nur sinnvolle Variabilitätsbeziehungen zwischen den verwendeten Komponenten identifiziert werden. Zudem ist es dadurch auch möglich, alle Eingabemodelle parallel zu verarbeiten, da wir die Anzahl an Komponenten reduzieren, die miteinander verglichen werden müssen, sodass der Aufwand für die Bestimmung der Variabilität sinkt.

Unser Clustering-Algorithmus ist darauf ausgerichtet, das vorhandene Wissen über die Struktur (z.B. Layer, Tiers und Kategorien) der zu analysierenden TAPs zu nutzen. Daher benötigen wir keine klassischen Data-Mining-Algorithmen[1], um geeignete

[1] beispielsweise Partitionierungs- (z.B. k-Means) oder Raster-basierte (z.B. STING) Clustering-Techniken [Ber06]

Cluster zu identifizieren. Durch die Verwendung von strukturellen Informationen, können wir sinnvolle Cluster mit kleinen vergleichbaren Teilmengen von Komponenten bilden und so unrealistische Beziehungen zwischen funktional unterschiedlichen Elementen herausfiltern. Hierzu machen wir die folgenden beiden Annahmen:

1) Die Variabilität von TAPs wird durch die Verwendung von Komponenten in konkreten Layer-Tier-Intersections bestimmt.

2) Komponenten mit unterschiedlichen Kategorien können keine Varianten sein, da sie für unterschiedliche Zwecke vorgesehen sind.

Beide Annahmen gelten für die Bestimmung der Variabilität von TAPs, da sowohl der Vergleich von Komponenten über verschiedene Layer-Tier-Intersections hinweg als auch der Vergleich von Komponenten mit verschiedenen Kategorien zu unrealistischen Variabilitätsbeziehungen führt. Beispielsweise könnte ein Vergleich der Komponente `MySQL 5.6` auf der `Daten-Server-Intersection` des `TAP1` (vgl. Tabelle 3.1) mit dem Element `Internet Explorer 8` auf der `Präsentation-Client-Intersection` des TAP4 (vgl. Tabelle 3.1) zur Erkennung von zwei alternativen Varianten führen, obwohl beide Komponenten funktional unterschiedlich und für den Einsatz auf verschiedenen Tiers vorgesehen sind.

Dies führt jedoch dazu, dass eventuell vorhandene Abhängigkeiten zwischen einzelnen Layer-Tier-Intersections nicht für die Variabilitätsbestimmung berücksichtigt werden. So wäre es beispielsweise denkbar, dass zwei unterschiedliche Konfigurationen oder Teilkonfigurationen als zwei einzelne Varianten betrachtet und miteinander verglichen werden könnten.

Definition 3.4: Konfiguration im Kontext von Technologiearchitekturen

In Anlehnung an die Definition von Seidl [Sei17], wird unter der *Konfiguration* im Kontext von Technologiearchitekturen eine gültige Teilmenge aller konfigurierbaren physischen Elemente verstanden, welche die technische Architektur eines Softwaresystems (TAP) bestimmen.

Definition 3.5: Teilkonfiguration im Kontext von Technologiearchitekturen

Als *Teilkonfiguration* wird eine Teilmenge einer gültigen Konfiguration verstanden, welche einen bestimmten Teil der technischen Architektur eines Softwaresystems bestimmt. Dieser Teil ist durch ausgewählte Intersections eines TAPs definiert.

Hierdurch wäre es möglich, nicht nur einzelne Komponenten im Hinblick auf unnötige Variabilität zu analysieren, sondern dies auch für verschiedenen Sets von kombinierten Elementen in Form von Konfigurationen oder Teilkonfigurationen zu ermöglichen. Allerdings hat sich in Gesprächen mit Experten ergeben, dass die Variabilitätsbestimmung für einzelne Komponenten einen deutlich höheren Mehrwert für die

Identifizierung und Reduzierung unnötiger Variabilität in TAPs bietet. Dies begründen die Experten vor allem damit, dass für einzelne Komponenten feingranularere Entscheidungen getroffen werden können und entsprechende Restrukturierungen so besser umsetzbar sind. Nichtsdestotrotz stellt die Berücksichtigung von Konfigurationen und Teilkonfigurationen eine interessante Erweiterung unseres Ansatzes im Rahmen von zukünftigen Arbeiten dar.

Clusteranalyse

Die *Clusteranalyse* für unseren Variabilitätsmining-Ansatz ist in zwei Schritte unterteilt und in der folgenden Abbildung 3.6 schematisch dargestellt.

Im ersten Schritt *Clustering nach Intersection* bilden wir *Grobcluster* von Komponenten, die sich, über alle betrachteten TAPs hinweg, auf derselben Layer-Tier-Intersection befinden. Hierfür werden zuerst alle verfügbaren Layer-Tier-Intersections identifiziert und jeweils ein Grobcluster für jede dieser Intersections gebildet. Anschließend werden alle Komponenten der betrachteten TAPs den entsprechenden Clustern anhand ihrer Layer-Tier-Intersection zugeordnet. Dabei verfügen die zu analysierenden TAPs nicht immer über die gleichen Layer und Tiers. So kann ein bestimmter TAP beispielsweise einen dedizierten Datenbank-Tier enthalten, wäh-

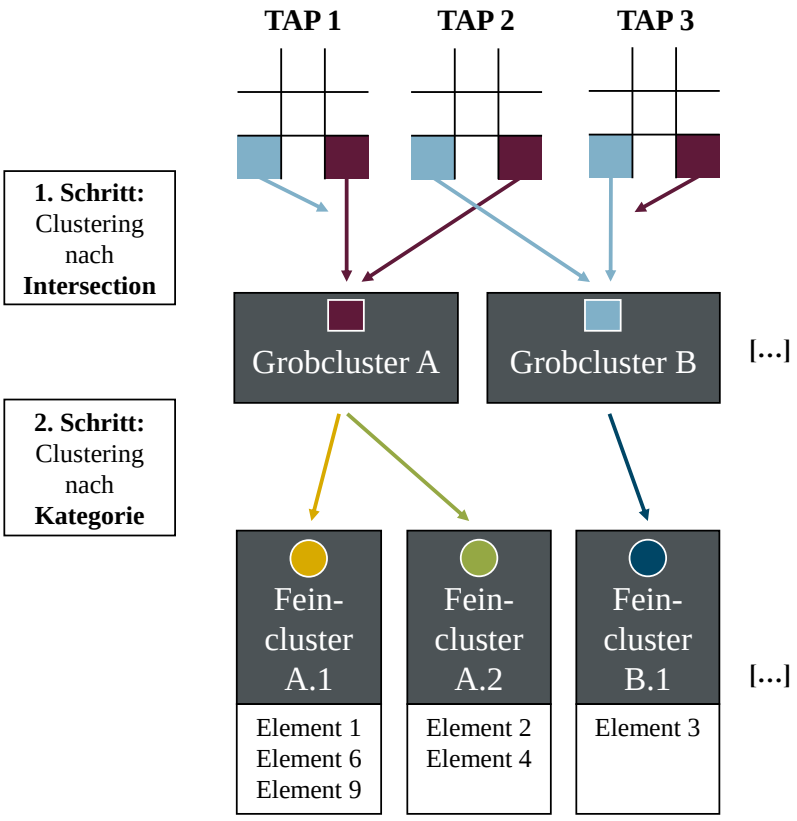


Abbildung 3.6: Schematische Darstellung der Clusteranalyse

rend alle anderen betrachteten TAPs diesen Tier nicht berücksichtigen. Dies hat auf die Clusteranalyse allerdings keine Auswirkung, da jede Komponente in einem TAP-Modell ihrem spezifischen Layer und Tier zugewiesen ist. Dieser Aspekt wurde bei der Entwicklung des zugrunde liegenden Metamodells berücksichtigt (vgl. Abbildung 3.3). So werden Komponenten dem gleichen Grobcluster nur dann zugeordnet, wenn sie auch im selben Layer und im gleichen Tier verbaut sind.

Wie in Abbildung 3.6 zu erkennen ist, werden so beispielsweise alle Komponenten aus der *rot* gekennzeichneten Intersection (z.B. `Hardware-Daten-Intersection`) der `TAPs 1-3` im *Grobcluster A* zusammengefasst und alle Elemente aus der *hellblau* gekennzeichneten Intersection (z.B. `Hardware-Client-Intersection`) im *Grobcluster B* gruppiert. Die Tabelle 3.5 zeigt die Bildung eines exemplarischen *Grobclusters A* für das fortlaufende Beispiel (Tabelle 3.1). In diesem Grobcluster werden alle Komponenten (z.B. `Tomcat` und `Tivoli Access Manager`) der vier TAPs gruppiert, die sich auf der `Applikation-Server-Intersection` befinden.

Da die Komponenten in solchen Grobclustern funktional sehr unterschiedlich sein können, ist es notwendig diese groben Cluster weiter zu verfeinern, um sinnvolle Variabilitätsbeziehungen identifizieren zu können. Hierzu machen wir uns im zweiten Schritt *Clustering nach Kategorie* das verwendete Kategoriensystem zu Nutze, indem wir *Feincluster* bilden, welche alle Komponenten eines Grobclusters mit derselben Kategorie zusammenfassen. So entstehen beispielsweise für den *Grobcluster A* die beiden *Feincluster A.1* und *A.2*, welche Komponenten mit der *gelben* (z.B. `x86-Server-Architektur`) beziehungsweise der *grünen* Kategorie (z.B. `Sparc-Architektur`) aufnehmen. Die so entstandenen Feincluster beinhalten demnach nur noch solche Komponenten, die eine funktionale Ähnlichkeit zueinander haben und dadurch sinnvolle Variabilitätsbeziehungen aufweisen können.

Die Tabelle 3.6 zeigt die *Feincluster A.1* und *A.2* für die `Applikation-Server-Intersection` des fortlaufenden Beispiels. Das *Grobcluster A* enthält dabei Komponenten, die entweder der Kategorie *Webserver* (z.B. `Tomcat`)

Grobcluster A: Applikation-Server-Intersection	
TAP	**Komponenten**
1	Tomcat 7 OpenLDAP 2.1
2	Tomcat 6
3	Websphere Appl. Server 7.0 Tivoli Access Manager 6.1
4	Tomcat 7

Tabelle 3.5: Grobcluster A für die Applikation-Server-Intersection des fortlaufenden Beispiels

Applikation-Server-Intersection		
TAP	**Feincluster A.1: Webserver**	**Feincluster A.2: Authentifizierung**
1	Tomcat 7	OpenLDAP 2.1
2	Tomcat 6	–
3	Websphere Appl. Server 7.0	Tivoli Access Manager 6.1
4	Tomcat 7	–

Tabelle 3.6: Feincluster A.1 und A.2 für die Applikation-Server-Intersection des fortlaufenden Beispiels

oder der Kategorie *Authentifizierung* (z.B. `OpenLDAP`) angehören. Somit kann der *Grobcluster A* in die beiden *Feincluster A.1* und *A.2* unterteilt werden, die so nur noch Komponenten mit derselben Kategorie enthalten.

Durch die Ausführung dieser beiden Clustering-Schritte identifizieren wir eine Gruppe von Feinclustern, die sich in derselben Layer-Tier-Intersection befinden und jeweils nur Komponenten desselben Typs enthalten. Hierdurch eliminieren wir unrealistische Variabilitätsbeziehungen für die betrachteten TAPs und reduzieren die erforderlichen Vergleiche zur Bestimmung dieser Variabilitätsbeziehungen. Sind zusätzliche Detailinformationen über die Eigenschaften (z.B. Funktionen) der betrachteten Komponenten verfügbar, können diese genutzt werden, um die Komponenten in den Feinclustern nach weiteren Kriterien zu gruppieren. Hierdurch entsteht eine größere Anzahl an Feincluster, welche eine jeweils geringere Menge von Komponenten beinhalten, wodurch eine noch genauere Analyse der Variabilitätsbeziehungen möglich wird. Allerdings sollte das Clustering nicht zu detailliert erfolgen, da die Cluster sonst nur wenige Komponenten beinhalten könnten und dadurch womöglich nicht alle Variabilitätsbeziehungen identifiziert werden.

Zum Abschluss der Clusteranalyse erstellen wir für jeden entstandenen Feincluster ein sogenanntes *Vergleichselement*, welches alle Komponenten des jeweiligen Feinclusters beinhaltet (z.B. `Tomcat 6`, `Tomcat 7` und `WebSphere App. Server 7.0` aus dem *Feincluster A.1*). Solche Vergleichselemente kapseln verwandte Komponenten während der Verarbeitung ein und werden in den nachfolgenden Phasen analysiert, um inhärente Variabilitätsbeziehungen zu identifizieren.

3.2.2 Splitting-Phase

Die in der Clusteranalyse (vgl. Unterabschnitt 3.2.1) entstandenen Feincluster schließen unrealistische Variabilitätsbeziehungen zwischen Komponenten mit unterschiedlichen Kategorien oder über verschiedene Layer und Tiers hinweg aus. Obwohl die generierten Vergleichselemente somit die Identifizierung korrekter Variabilitätsbeziehungen zulassen, entsprechen sie möglicherweise nicht immer

den Erwartungen von Domänenexperten. So könnten zum Beispiel die beiden physischen Elemente `Tomcat 6` und `Tomcat 7` zwei unabhängige *Konfigurationsoptionen* darstellen, obwohl sie dem gleichen logischen Element `Tomcat` zugeordnet sind.

Definition 3.6: Konfigurationsoption

Unter einer *Konfigurationsoption* wird eine Komponente auf Ebene eines logischen Elementes verstanden, welche in einer spezifischen Layer-Tier-Intersection verwendet wird und sich durch eine explizite Variabilitätsbeziehung auszeichnet. Diese Konfigurationsoption kann in einem TAP durch unterschiedliche physische Elemente (Versionen), die dem logischen Element zugeordnet sind, realisiert werden.

Um solche Erwartungen von Experten berücksichtigen zu können, wird in dieser *Splitting-Phase* ein regelbasiertes System verwendet, welches unerwünschte Variabilitätsbeziehungen zwischen verwandten Komponenten mit Hilfe von Regeln eliminiert. Hierfür werden die Informationen über die logischen Elemente der jeweiligen Komponenten ausgewertet und die Vergleichselemente anhand dessen aufgesplittet. Die folgende Abbildung 3.7 stellt dieses Vorgehen schematisch dar. Hier wird das Vergleichselement für den *Feincluster A.1* nach Regelanwendung in die beiden *Vergleichselemente A.1.1* (für Komponenten mit dem *logischen Element A*) und *A.1.2* (für Komponenten mit dem *logischen Element B*) aufgesplittet.

In den folgenden Tabellen 3.7 und 3.8 sind zwei *Szenarien A* und *B* (in Anlehnung an das fortlaufende Beispiel aus Tabelle 3.1) beschrieben, die wir bei der Arbeit

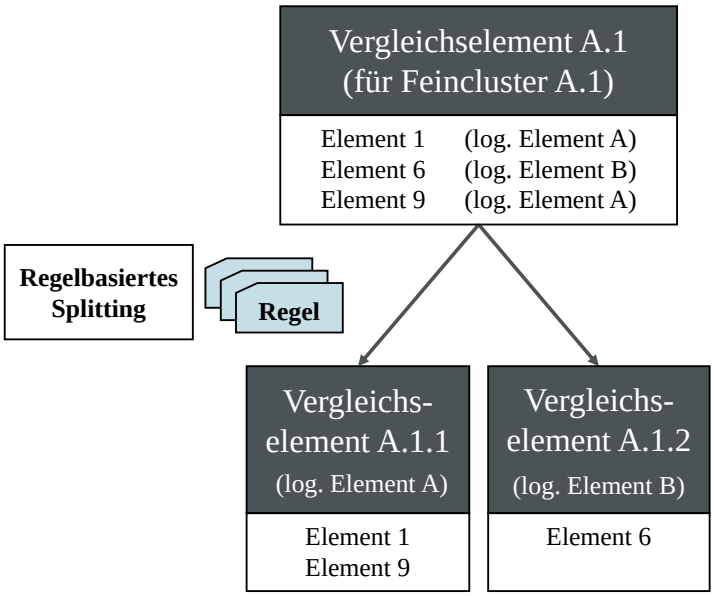


Abbildung 3.7: Schematische Darstellung des Splittings von Vergleichselementen

mit unserem Industriepartner identifiziert haben. Jede Tabelle repräsentiert genau ein beispielhaftes Vergleichselement mit gleichartigen Komponenten aus verschiedenen TAPs. Für jedes Szenario sind die beteiligten TAPs, die enthaltenen logischen und physischen Elemente sowie das durch unseren Ansatz ermittelte und das von Experten erwartete Ergebnis der Variabilitätsbestimmung dargestellt. Eine nähere Beschreibung der Variabilitätsbeziehungen für Komponenten in TAPs erfolgt dabei in der nachfolgenden Merging-Phase (vgl. Unterabschnitt 3.2.3). Damit unser Ansatz auch die von Experten erwarteten Ergebnisse erzielen kann, werden die zwei nachfolgenden Regeln angewendet, mit deren Hilfe die Vergleichselemente der beiden Szenarien analysiert und aufgesplittet werden können.

Regel 1 – Extraktion verpflichtender Komponenten

Die *Regel 1* splittet ein bestehendes Vergleichselement, welches alternative Komponenten beinhalten, anhand von logischen Elementen auf, wenn dabei mindestens eine Komponente des gleichen logischen Elementes in jedem der analysierten TAPs eingesetzt wird.

Zur Verdeutlichung dieser Regel wird in Tabelle 3.7 das *Szenario A* mit unterschiedlichen Browser-Komponenten aus zwei verschiedenen TAPs beschrieben, die sich alle in der `Präsentation-Client-Intersection` befinden. Unser Algorithmus würde in diesem Szenario drei alternative Elemente (`A1-A3`) identifizieren, da in beiden TAPs ein `Firefox` in verschiedenen Versionen (`42` und `49`) eingesetzt wird und lediglich der `Internet Explorer` im `TAP2` als Browser zur Anwendung kommt. Domänenexperten würden allerdings in diesem Fall erwarten, dass der `Firefox` als *verpflichtende* Komponente für die zwei analysierten TAPs identifiziert wird, um zu verdeutlichen, dass diese Komponente in allen Varianten implementiert ist. Somit repräsentiert der `Internet Explorer` eine unabhängige Konfigurationsoption in diesem Beispiel. Durch Analyse der logischen Elemente, die den beteiligten Komponenten zugeordnet sind, lässt sich feststellen, dass der `Firefox` unabhängig von seiner Version in allen TAPs vorkommt und dass der `Internet Explorer` keine Beziehung zu dem logischen Element des Firefox' hat (d.h. sie haben unterschiedliche logische Elemente). Somit kann das Vergleichselement in zwei separate Vergleichselemente aufgeteilt werden, die entweder die verpflichtende (`V1`) Komponente `Firefox` oder den optionalen (`O1`) `Internet Explorer` beinhalten.

TAP	Phy. Element	Log. Element	Ergebnis	Erwartetes Ergebnis
1	Firefox 42	Firefox	A1	V1
2	Firefox 49	Firefox	A2	V1
2	Internet Explorer 8	Internet Explorer	A3	O1

V = verpflichtend *A* = alternativ *O* = optional

Tabelle 3.7: Szenario A für Regel 1

Regel 2 – Extraktion unabhängiger Konfigurationsoptionen

Die *Regel 2* splittet ein bestehendes Vergleichselement mit optionalen Komponenten anhand ihrer logischen Elemente auf, wenn die optionalen Komponenten unabhängig von einander sind.

Zur Erklärung dieser Regel wird in Tabelle 3.8 das *Szenario B* dargestellt, welches ein Vergleichselement für Datenbank-Komponenten auf der `Daten-Server-Intersection` von drei verschiedenen TAPs betrachtet. Wie hier zu erkennen ist, wird die Komponente `MySQL 5.6` im `TAP1` und die Komponente `IBM DB2 10` im `TAP3` eingesetzt. `TAP2` hingegen verfügt über keine Komponente in der betrachten Intersection. Unser Verfahren würde diese Datenbank-Komponenten als eine einzige Konfigurationsoption (`O1`) für die drei TAPs identifizieren, da sie vom selben Typ sind (sie entsprechen derselben Kategorie *RDBS*) und der `TAP2` keine Datenbank enthält. Die Domänenexperten unseres Industriepartners argumentieren jedoch, dass die Hervorhebung der beiden Datenbank-Komponenten als unabhängige Konfigurationsoptionen wünschenswert ist, da sie in unterschiedlichen TAPs und nicht zusammen vorkommen. Durch die Analyse ihrer entsprechenden logischen Elemente, lässt sich feststellen, dass beide Komponenten unabhängig voneinander sind (d.h. sie sind unterschiedlichen logischen Elementen zugeordnet). Somit können sie auch als unabhängige Konfigurationsoptionen (`O1` und `O2`) dargestellt und in zwei separate Vergleichselemente aufgeteilt werden. Da diese beiden Optionen nicht in der selben Konfiguration verwendet werden können, werden sie in einer späteren Phase zu einer *Relationsgruppe* zusammengefasst (vgl. Unterabschnitt 3.3.1).

Diese beiden Regeln sind gemeinsam mit den Experten unseres Industriepartners entwickelt worden, nachdem die Evaluation des vorgeschlagenen Variabilitätsmining-Ansatzes vereinzelt zu unerwarteten Ergebnissen geführt hat (vgl. Unterabschnitt 6.2.2). So ist es nun mit Hilfe dieser Regeln in der nächsten Phase *Merging* (vgl. Unterabschnitt 3.2.3) möglich, die konkrete Variabilität von TAPs zu bestimmen, ohne dabei Variabilitätsbeziehungen zwischen einzelnen, verwandten Komponenten mit demselben logischen Element zu identifizieren. Während der Ausführung dieser Regeln iteriert unser Algorithmus dabei solange über die Liste der erzeugten Vergleichselemente (vgl. Unterabschnitt 3.2.1), bis keine weiteren Optimierungen mehr gefunden werden. Darüber hinaus gehende Splitting-Regeln sind für eine korrekte Variabilitätsbestimmung von TAPs nicht erforderlich.

TAP	Phy. Element	Log. Element	Ergebnis	Erwartetes Ergebnis
1	MySQL 5.6	MySQL	O1	O1
2	–	–	–	–
3	IBM DB2 10	IBM DB2	O1	O2

Tabelle 3.8: Szenario B für Regel 2

3.2.3 Merging-Phase

Nachdem die Vergleichselemente in der letzten Phase regelbasiert aufgesplittet wurden, um zu vermeiden, dass von Experten unerwartete Variabilitätsbeziehungen entstehen, können wir nun in dieser *Merging-Phase* konkrete Variabilitätsbeziehungen zwischen allen Komponenten aus den analysierten TAPs identifizieren und anschließend in ein gemeinsames 150%-Modell zusammenführen. Auf Basis dieser aggregierten Variabilitätsinformationen ist es Domänenexperten möglich, geeignete Analysen durchzuführen und fundierte Architekturentscheidungen zu treffen. So müssen sich die Experten nicht mehr auf potentiell fehlerhafte Ergebnisse aus einer zeitraubenden manuellen Analyse von betrachteten TAPs verlassen, sondern können ihre Architekturarbeit auf zuverlässige Fakten durch automatisch generierte 150%-Modelle stützen.

Identifizierung von Variabilitätsbeziehungen

Als erstes muss die Variabilität jedes Vergleichselementes klassifiziert werden. Diese Klassifizierung ist notwendig, um die entsprechenden Variabilitätsbeziehungen für die Domänenexperten explizit zu machen. Für die Klassifizierung wenden wir die folgenden Regeln an:

verpflichtend: Alle Komponenten des Vergleichselements teilen sich *dasselbe logische Element*, und es existiert *kein TAP*, der nicht durch eine Komponente im Vergleichselement repräsentiert wird.

optional: Alle Komponenten des Vergleichselements teilen sich *dasselbe logische Element*, und es existiert *mindestens ein TAP*, der nicht durch eine Komponente im Vergleichselement repräsentiert wird.

alternativ: Das Vergleichselement ist weder *verpflichtend* noch *optional*, daher beinhaltet es eine Komponente für jeden betrachteten TAP, wobei die jeweiligen Komponenten mindestens zwei verschiedenen logischen Elementen zugeordnet sein müssen.

In der nachfolgenden Tabelle 3.9 werden die beschriebenen Variabilitätsbeziehungen noch einmal anhand des fortlaufenden Beispiels (vgl. Tabelle 3.1) erklärt.

Variabilität	TAP1	TAP2	TAP3	TAP4
verpflichtend (V)	Windows 7	Windows 10	Windows 7	Windows 7
optional (O)	phpMyAdmin 4.6	–	–	phpMyAdmin 2.5
alternativ (A)	Redhat 6	CentOS 6	Redhat 6	Redhat 7

Tabelle 3.9: Mögliche Variabilitätsbeziehungen zwischen Komponenten in verschiedenen TAPs

Wie in Tabelle 3.9 zu erkennen ist, befindet sich in jedem der vier TAPs eine Komponente, die dem logischen Element `Windows` zugeordnet ist. Zwar sind hier unterschiedliche physische Elemente (`Windows 7` und `Windows 10`) verbaut, jedoch gehören sie dem gleichen logischen Element an. Somit kann zwischen den Windows-Komponenten eine *verpflichtend*-Beziehung identifiziert werden. Darüber hinaus sind in diesem Beispiel die Komponenten `phpMyAdmin 4.6` in `TAP1` und `phpMyAdmin 2.5` in `TAP4` verbaut. Da auch diese beiden Komponenten dem gleichen logischen Element `phpMyAdmin` angehören und zudem in `TAP2` und `TAP3` keine anderen, gleichartigen Komponenten in dieser Intersection verbaut sind, kann hierfür eine *optional*-Beziehung bestimmt werden. Zudem sind in allen vier TAPs Betriebssystem-Komponenten auf der `Betriebssystem-Server-Intersection` implementiert. Hierbei handelt es sich um die beiden physischen Komponenten `Redhat 6` und `Redhat 7`, die dem logischen Element `Redhat` zugeordnet sind, sowie um die physische Komponente `CentOS 6`, die mit dem logischen Element `CentOS` verbunden ist. Aufgrund der Tatsache, dass hierbei in allen TAPs gleichartige Komponenten zum Einsatz kommen, welche aber zwei unterschiedlichen logischen Elementen zugeordnet sind, kann zwischen diesen Komponenten eine *alternativ*-Beziehung identifiziert werden.

Zusammenführung aller Variabilitätsinformationen

Nach Identifizierung der einzelnen Variabilitätsbeziehungen verschmelzen wir alle Komponenten aus den verarbeiteten Vergleichselementen zu einem einzigen Modell und annotieren deren Variabilität entsprechend ihrer Klassifikation sowie deren TAPs, aus denen sie stammen. Die folgende Abbildung 3.8 zeigt dieses Vorgehen schematisch.

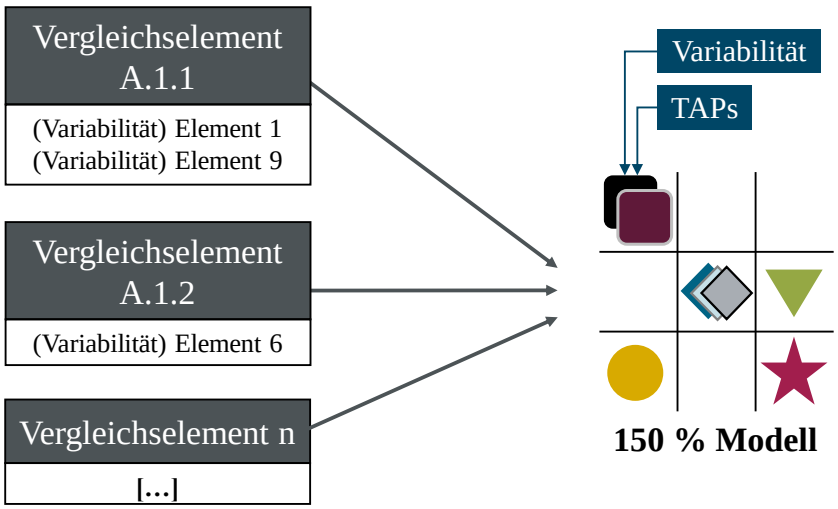


Abbildung 3.8: Schematische Darstellung des Mergings von Vergleichselementen

(Var.) log. Elem.	TAP1	TAP2	TAP3	TAP4
(V1) Windows	Windows 7	Windows 10	Windows 7	Windows 7
(O1) phpMyAdmin	phpMyAdm. 4.6	–	–	phpMyAdm. 2.5
(A1) Redhat	Redhat 6	–	Redhat 6	Redhat 7
(A2) CentOS	–	CentOS 6	–	–

Var. = Variabilität *V* = verpflichtend *O* = optional *A* = alternativ

Tabelle 3.10: Zusammengefasste Variabilitätsbeziehungen für logische Elemente

Um das resultierende 150%-Modell nicht mit Informationen zu überfrachten und eine leichtere manuelle Analyse der Variabilität durch Domänenexperten zu ermöglichen, werden Variabilitätsbeziehungen zwischen einzelnen physischen Elementen, die dem gleichen logischen Element angehören, zu einer *Variabilitätsinformation* für die entsprechende Konfigurationsoption zusammengefasst. Die Tabelle 3.10 verdeutlicht dies anhand der identifizierten Variabilitätsbeziehungen aus der Tabelle 3.9.

Wie hier zu erkennen ist, sind die Variabilitätsbeziehungen zwischen den einzelnen physischen Elemente für `Windows` und `phpMyAdmin` zu zwei konkreten Variabilitätsinformationen zusammengefasst (`V1` und `O1`), da sie jeweils dem gleichen logischen Element zugeordnet sind. Für die alternativen Komponenten werden allerdings aufgrund ihrer unterschiedlichen logischen Elemente (`Redhat` und `CentOS`) auch verschiedene Variabilitätsinformationen benötigt (`A1` und `A2`). Um die Beziehung zwischen diesen beiden Alternativen weiterhin zu verdeutlichen, werden sie in einer spezifischen *Alternativgruppe (AG)* zusammengefasst. In diesem Beispiel wird also die Alternativgruppe `AG1((A1) Redhat, (A2) CentOS))` gebildet.

Neben der Variabilität ermitteln wir auch die *Häufigkeit H* der Nutzung von verschiedenen Konfigurationsoptionen sowie ihren enthalten physischen Komponenten und fügen diese Ergebnisse ebenfalls dem 150%-Modell hinzu. Dabei meint eine Konfigurationsoption $ko_a = ko(v_i, l_e)$ die Nutzung eines bestimmten logischen Elementes l_e mit einer spezifischen Variabilitätsinformation v_i (z.B. `(O1) phpMyAdmin`). Deren Nutzungshäufigkeit zeigt an, wie oft diese Konfigurationsoption über alle betrachteten TAPs hinweg verwendet wird und lässt sich mit Hilfe der folgenden Gleichung 3.2 ermitteln:

$$H_{ko_a} = H_{ko(v_i,l_e)} = \frac{N_T \; mit \; ko(v_i, l_e)}{N_T} \tag{3.2}$$

So ergibt sich die Häufigkeit H_{ko_a} aus dem Verhältnis der Anzahl N_T an TAPs mit der spezifischen Konfigurationsoption $ko_a = ko(v_i, l_e)$ zu der Anzahl N_T aller betrachteten TAPs.

Ergänzend dazu stellt die Nutzungshäufigkeit einer bestimmten Komponente k_j (z.B. `phpMyAdmin 4.6`) ihre Verwendungshäufigkeit im Rahmen einer spezifischen Konfigurationsoption ko_a dar. So lässt sich feststellen, wie oft eine bestimmte Version beziehungsweise ein bestimmtes physisches Element zur Realisierung einer

Konfigurationsoption zum Einsatz kommt. Dies lässt sich durch die folgende Gleichung 3.3 ermitteln:

$$H_{k_j(ko_a)} = \frac{N_T \; mit \; k_j(ko_a)}{N_T \; mit \; ko_a} \tag{3.3}$$

Demnach wird die Häufigkeit $H_{k_j(ko_a)}$ ermittelt, indem das Verhältnis der Anzahl N_T an TAPs mit k_j von ko_a zu der Anzahl N_T an TAPs mit der spezifischen Konfigurationsoption ko_a betrachtet wird.

In der folgenden Tabelle 3.11 sind exemplarische Nutzungshäufigkeiten für zwei Konfigurationsoptionen aus dem fortlaufenden Beispiel (vgl. Tabelle 3.1) dargestellt. Für die Konfigurationsoption `(V1) Windows` kann eine Häufigkeit von H_{ko_a}=100% ermittelt werden, da diese in allen TAPs verwendet wird. Demzufolge errechnet sich für die Komponente `Windows 7` eine Häufigkeit von $H_{kj(ko_a)}$=75% sowie für die Komponente `Windows 10` von $H_{kj(ko_a)}$=25%. Im Fall der Konfigurationsoption `(O1) phpMyAdmin` kann die Häufigkeit mit H_{ko_a}=50% bestimmt werden, weil diese in zwei von vier TAPs eingesetzt wird. Da hierbei die Komponenten `phpMyAdmin 2.5` und `phpMyAdmin 4.6` in jeweils einem dieser beiden TAPs verwendet werden, ergibt sich für beide eine Häufigkeiten von $H_{kj(ko_a)}$=50%.

Die folgende Tabelle 3.12 stellt das 150%-Modell für unser fortlaufendes Beispiel aus Tabelle 3.1 dar. Aus Gründen der Lesbarkeit zeigt diese Visualisierung nicht die konkreten TAP-Bezeichnungen, in denen die einzelnen Komponenten verbaut sind. Wie hier zu sehen ist, hat unser Verfahren die *verpflichtenden* Konfigurationsoptionen `(V1) Firefox`, `(V2) Windows`, `(V3) x86` und `(V4) x86-Server` korrekt erkannt. Außerdem wurden die *alternativen* Webserver-Komponenten in `AG1((A1) Tomcat, (A2) WebSphere App. Server)` und die *alternativen* Server-Betriebssysteme in `AG2((A3) Redhat, (A4) CentOS)` richtig identifiziert. Darüber hinaus sind auch die 6 optionalen Konfigurationsoptionen `(O1-O6)` korrekt bestimmt worden. In diesem Zusammenhang ist beispielsweise auch richtig erkannt worden, dass die Konfigurationsoption `(O1) phpMyAdmin` in zwei von vier analysierten TAPs verbaut ist und die beiden eingesetzten Versionen jeweils einmal vorkommen.

Durch die Verwendung unseres automatischen Mining-Verfahrens können Architekten eine großen Anzahl von TAPs analysieren und deren Variabilität bestimmen, ohne

Konfigurationsoption	**TAP**	**H_{ko_a}**	**Komponente**	**$H_{kj(ko_a)}$**	**TAP**
(V1) Windows	1–4	100%	Windows 7	75%	1,3,4
			Windows 10	25%	2
(O1) phpMyAdmin	1,4	50%	phpMyAdmin 2.5	50%	4
			phpMyAdmin 4.6	50%	1

Tabelle 3.11: Beispielhafte Nutzungshäufigkeiten für Konfigurationsoptionen und deren physische Komponenten

	Client			Server		
	Var.	**(H_{ko_a}) ko_a**	**k_j (H_{k_j})**	**Var.**	**(H_{ko_a}) ko_a**	**k_j (H_{k_j})**
Präsentation	V1	(4/4) Firefox	42 (3/4)	O1	(2/4) phpMyAdmin	2.5 (1/2)
			49 (1/4)			4.6 (1/2)
	O2	(1/4) Internet Explorer	8 (1/1)			
Applikation				A1	(3/4) Tomcat	7 (2/3)
				AG1		6 (1/3)
				A2	(1/4) WebSphere App. Server	7.0 (1/1)
				O3	(1/4) OpenLDAP	2.1 (1/1)
				O4	(1/4) Tivoli Access Manager	6.1 (1/1)
Daten				O5	(2/4) MySQL	5.7 (1/2)
						5.6 (1/2)
				O6	(1/4) IBM DB2	10 (1/1)
Betriebssystem	V2	(4/4) Windows	7 (3/4)	A3	(3/4) Redhat	6 (2/3)
			10 (1/4)	AG2		7 (1/3)
				A4	(1/4) CentOS	6 (1/1)
Hardware	V3	(4/4) x86		V4	(4/4) x86-Server	

Var. = Variabilität *V* = verpflichtend *A* = alternativ *O* = optional *H* = Häufigkeit *ko* = Konfigurationsoption *k* = Komponente

Tabelle 3.12: 150% Modell für das fortlaufende Beispiel aus Tabelle 3.1

dabei wertvolle Zeit für die manuelle Analyse dieser TAPs zu verschwenden. Das Ergebnis unseres Verfahrens in Form eines generierten 150%-Modells kann dann beispielsweise dafür verwendet werden, Maßnahmen zur Reduzierung der identifizierten Variabilität abzuleiten (vgl. Kapitel 4), um so Kosten für die Wartung und Weiterentwicklung der betrachteten TAPs einzusparen. Die ermittelte Variabilität zeigt dabei Restrukturierungspotenziale auf, die zuvor nicht offensichtlich waren. Durch Anreicherung dieser Informationen mit dem Domänenwissen von Experten können begründete und objektiv nachvollziehbare Entscheidungen zur Restrukturierung getroffen werden. Hierdurch lassen sich zum Beispiel auch veraltete Komponenten sowie technische Schulden aus Altprojekten erkennen und beseitigen.

3.3 Kandidaten für unnötige Variabilität

Ein 150%-Modell beschreibt die Variabilität für eine bestimmte Menge von TAPs und visualisiert damit alle eingesetzten Varianten für einen bestimmten Anwendungszweck, wie zum Beispiel ein konkreter Architekturtyp oder eine implementierte Technologie. Allerdings verursacht nicht jede Komponente, die in einem TAP verbaut ist, *unnötige Variabilität*.

Definition 3.7: Unnötige Variabilität

Wir definieren *unnötige Variabilität* als die Menge an Konfigurationsoptionen, die für den betrachteten Anwendungszweck von den analysierten TAPs nicht benötigt wird. Dabei ergibt sich die Notwendigkeit einer Konfigurationsoption entweder durch betriebliche oder durch technische Anforderungen.

Solch eine unnötige Variabilität kann beispielsweise vorliegen, wenn zwei unterschiedliche Betriebssystem-Konfigurationsoptionen (z.B. `(A1) Redhat Linux` und `(A2) SuSe Linux`) zum Einsatz kommen, um darauf den gleichen Applikationsserver für Java-Anwendungen zu betreiben (z.B. `Tomcat`), obwohl weder technische noch betriebliche Anforderungen ein zweites Betriebssystem erfordern. Somit kann eine der beiden Betriebssystem-Komponenten als unnötige Variabilität eingestuft werden. Da zur konkreten Bestimmung allerdings weitere Analysen erforderlich sind (vgl. Kapitel 4), werden zunächst alle möglichen Konfigurationsoptionen als *Kandidaten für unnötige Variabilität* betrachtet.

Definition 3.8: Kandidat für unnötige Variabilität

Als *Kandidat für unnötige Variabilität* (Kurzform *Kandidat*) wird eine Konfigurationsoption verstanden, die eine Variabilitätsbeziehung vom Typ *optional* oder *alternativ* aufweist oder eine *funktionale Beziehung* zu einer anderen Konfigurationsoption hat.

Zur Bestimmung von Kandidaten für unnötige Variabilität liefert das 150%-Modell von analysierten TAPs eine geeignete Grundlage, da es aggregierte Informationen über alle Variabilitätsbeziehungen enthält. Allerdings können zwischen einzelnen Konfigurationsoptionen, beziehungsweise den darin enthaltenen Komponenten, auch funktionale Beziehungen bestehen, welche durch das 150%-Modell bisher nicht betrachtet werden.

Definition 3.9: Funktionale Beziehung

Als *funktionale Beziehung* wird eine Beziehung zwischen zwei einzelnen Konfigurationsoptionen verstanden, deren jeweiliges logisches Element dem gleichen Komponententyp entspricht.

Dabei wird der *Komponententyp* maßgeblich durch die Kategorie einer Komponente bestimmt, kann allerdings auch weitere Informationen über deren Funktionen berücksichtigen, falls solche Daten zur automatisierten Auswertung vorliegen. So können beispielsweise zwei voneinander unabhängige, optionale Konfigurationsoptionen `(O1) F5-Big-IP` und `(O2) Citrix NetScaler` derselben Kategorie *Load Balancer* zugeordnet sein, da beide sehr ähnliche Funktionen für die Lastverteilung in Softwaresystemen bereitstellen. Aufgrund solcher ähnlichen Merkmale können Konfigurationsoptionen in funktionaler Beziehungen zueinander stehen, welche für die Identifizierung von Kandidaten berücksichtigt werden müssen.

Daher wird im nachfolgenden Unterabschnitt 3.3.1 zunächst näher beschrieben, wie funktionale Beziehungen zwischen Konfigurationsoptionen in betrachteten TAPs identifiziert und darauf aufbauend im Unterabschnitt 3.3.2 Kandidaten für unnötige Variabilität bestimmt werden können. Zudem wird im Unterabschnitt 3.3.3 gezeigt, wie die durch solche Kandidaten verursachte Variabilität gemessen werden kann.

3.3.1 Bestimmung von funktionalen Beziehungen

Informationen über funktionale Beziehungen zwischen einzelnen Konfigurationsoptionen werden für die Identifizierung von Kandidaten für unnötige Variabilität benötigt, da sie für die spätere Entscheidungsfindung zur Restrukturierung von TAPs erforderlich sind. Solche funktionalen Beziehungen können wie folgt bestimmt werden.

Im ersten Schritt *Cross Check* wird für jede Konfigurationsoption überprüft, ob es eine oder mehrere andere Konfigurationsoptionen mit Komponenten gibt, die dem gleichen Komponententyp entsprechen. Wenn dies der Fall ist, kann angenommen werden, dass die Komponenten dieser Konfigurationsoptionen sehr ähnliche Anforderungen erfüllen können und somit als *funktionale Alternativen* betrachtet werden können. Im Falle einer Übereinstimmung des Komponententyps wird daher eine *funktionale Relation (R)* zwischen den entsprechenden Konfigurationsoptionen erstellt.

In unserem fortlaufenden Beispiel kann eine solche funktionale Beziehung zwischen den Konfigurationsoptionen `(O3) OpenLDAP` und `(O4) Tivoli Access Manager` erkannt werden, da sie der gleichen Kategorie *Authentifizierung* entsprechen. Somit wird die Relation `R1((O3) OpenLDAP, (O4) Tivoli Access Manager)` erstellt.

Im zweiten Schritt *Bildung von Relationsgruppen* werden für die identifizierten Relationen spezifische Gruppen gebildet, welche alle gleichartige Konfigurationsoptionen in derselben Intersection zusammenfassen.

Definition 3.10: Relationsgruppe

Als *Relationsgruppe (RG)* wird eine Gruppe von Konfigurationsoptionen bezeichnet, welche die gleiche funktionale Beziehung aufweisen und sich auf derselben Intersection der betrachteten TAPs befinden.

Solche Relationsgruppen dienen dazu, alle Komponenten vom gleichen Typ innerhalb einer Intersection als funktionale Alternativen betrachten zu können und zwar unabhängig von ihrer ursprünglichen Variabilität.

In der nachfolgenden Tabelle 3.13 sind die Relationsgruppen für unser fortlaufendes Beispiel (vgl. Tabelle 3.1) dargestellt. Hierbei ist für die bereits beschriebene Relation `R1` die Relationsgruppe `RG1((O3) OpenLDAP, (O4) Tivoli Access Manager)` gebildet worden. Zusätzlich sind zwei weitere funktionale Beziehungen identifiziert und hierfür die beiden Relationsgruppen `RG2((V1) Firefox, (O1) Internet Explorer)` und `RG3((O5) MySQL, (O6) IBM DB2)` generiert worden.

Die so erstellten Relationsgruppen werden anschließend dem generierten 150%-Modell hinzugefügt, sodass sie für weitere Analysen zur Verfügung zu stehen und auch für Domänenexperten sichtbar sind.

Variabilität		TAP1	TAP2	TAP3	TAP4
RG1	O3	OpenLDAP 2.1	–	–	–
	O4	–	–	Tivoli Acc. Ma. 6.1	–
RG2	V1	Firefox 42	Firefox 42	Firefox 49	Firefox 42
	O1	–	–	–	Internet Explorer 8
RG3	O5	MySQL 5.6	–	–	MySQL 5.7
	O6	–	–	IBM DB2 10	–

RG = Relationsgruppe *V* = verpflichtend *O* = optional

Tabelle 3.13: Identifizierte Relationsgruppen für das fortlaufende Beispiel

3.3.2 Identifizierung von Kandidaten für unnötige Variabilität

Nachdem alle funktionalen Beziehungen identifiziert worden sind, können im nächsten Schritt die Kandidaten für unnötige Variabilität der analysierten TAPs ermittelt werden. Hierzu bestimmen wir alle Konfigurationsoptionen und deren verbaute Komponenten, die als Kandidat für unnötige Variabilität in Betracht kommen. Die folgende Abbildung 3.9 zeigt, welche Typen hierfür berücksichtigt werden.

Wie in der Abbildung 3.9 zu erkennen ist, werden zum einen alle unabhängigen optionalen Konfigurationsoptionen und zum anderen solche Konfigurationsoptionen, die einer Variabilitätsgruppen zugeordnet sind, als Kandidaten für unnötige Variabilität betrachtet. Dabei fassen wir unter einer *Variabilitätsgruppe* die Alternativgruppe und die Relationsgruppe zusammen. Während eine Alternativgruppe ausschließlich Konfigurationsoptionen mit Komponenten enthält, die eine *alternative* Variabilitätsbeziehung zueinander haben, besteht eine Relationsgruppe aus Konfigurationsoptionen, welche dieselbe *funktionale Beziehung* aufweisen. Solch eine Relationsgruppe kann dabei aus optionalen und aus verpflichtenden Konfigurationsoptionen bestehen. Jede dieser beiden Variabilitätsgruppen beinhaltet mindestens zwei Konfigurationsoptionen, welche jeweils in der Lage sind, die Anforderungen im Rahmen des allgemeinen Verwendungszwecks (vgl. Unterabschnitt 3.1.1) der ausgewählten TAPs zu erfüllen. Daher wäre es potentiell möglich, lediglich eine von ihnen zu verwenden, sodass alle Konfigurationsoptionen innerhalb solch einer Gruppe zunächst als Kandidaten für unnötige Variabilität identifiziert werden.

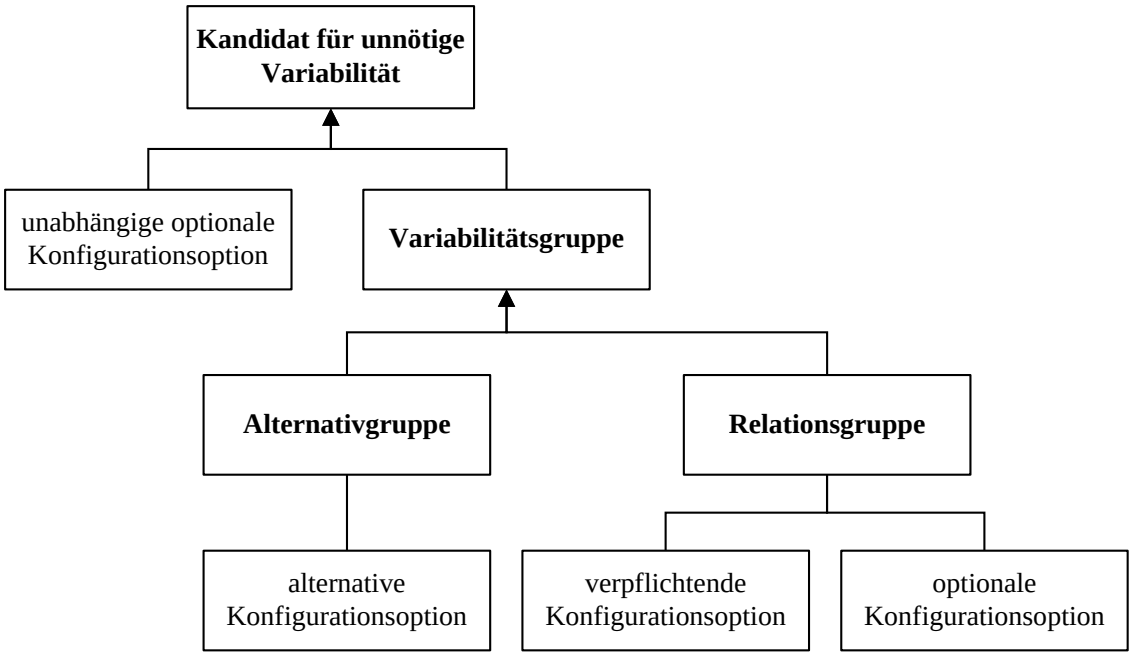


Abbildung 3.9: Typen von Kandidaten für unnötige Variabilität

Darüber hinaus werden unabhängige Konfigurationsoptionen mit optionalen Komponenten ebenfalls als Kandidaten betrachtet. Hierbei meint *unabhängig*, dass eine bestimmte Konfigurationsoption weder eine Variabilitätsbeziehung noch eine funktionale Beziehung zu einer anderen Konfigurationsoption aufweist. Da hiervon betroffene Komponenten nur in vereinzelten TAPs eingesetzt werden, sind sie möglicherweise nicht direkt für den Verwendungszweck der analysierten TAPs erforderlich. Dies stellt ein Anzeichen für sogenannte *Insellösungen* dar, die als eine Art Einzelanfertigung für Softwaresysteme angesehen werden können und die es aufgrund ihrer hohen Kosten für Wartung und Weiterentwicklung zu minimieren gilt. Somit werden solche Konfigurationsoptionen auch als Kandidaten für unnötige Variabilität identifiziert, obgleich sie auch einen individuellen Verwendungszweck erfüllen und damit für die Gesamtarchitektur erforderlich sein könnten. Dies kann allerdings nur durch zusätzliches Domänenwissen von Experten verifiziert werden, was in einer späteren Phase des Gesamtansatzes (vgl. Kapitel 5) ermöglicht wird.

In der folgenden Tabelle 3.14 sind die identifizierten Kandidaten für unser fortlaufendes Beispiel (vgl. Tabelle 3.1) aufgelistet. Hierzu zählen neben den bereits erwähnten Relationsgruppen `RG1, RG2` und `RG3` auch die Alternativgruppen `AG1` und `AG2` sowie die unabhängige Konfigurationsoption `(O1) phpMyAdmin`.

Die so identifizierten Kandidaten für unnötige Variabilität werden in der nächsten Phase des Gesamtansatzes (vgl. Kapitel 4) weiterverarbeitet, um geeignete Restrukturierungsempfehlungen für die analysierten TAPs abzuleiten, welche zur Identifizierung und Reduzierung der unnötigen Variabilität beitragen können.

Variabilität		**TAP1**	**TAP2**	**TAP3**	**TAP4**
RG1	O3	OpenLDAP 2.1	–	–	–
	O4	–	–	Tivoli Acc. Ma. 6.1	–
RG2	V1	Firefox 42	Firefox 42	Firefox 49	Firefox 42
	O1	–	–	–	Internet Expl. 8
RG3	O5	MySQL 5.6	–	–	MySQL 5.7
	O6	–	–	IBM DB2 10	–
AG1	A1	Tomcat 7	Tomcat 6	–	Tomcat 7
	A2	–	–	WebSph. Server 7.0	–
AG2	A3	Redhat 6	–	Redhat 6	Redhat 7
	A4	–	CentOS 6	–	–
–	O1	phpMyAdm. 4.6	–	–	phpMyAdm. 2.5

RG = Relationsgruppe *AG* = Alternativgruppe *V* = verpflichtend, *O* = optional, *A* = alternativ

Tabelle 3.14: Kandidaten für unnötige Variabilität des fortlaufenden Beispiels

3.3.3 Messung der Variabilität von identifizierten Kandidaten

Zur quantitativen Bestimmung der Variabilität, welche durch die identifizierten Kandidaten im generierten 150%-Modell *M1* verursacht wird, wenden wir die folgende Gleichung 3.4 an.

$$Variabilität(M_1) \;=\; N_{AG} \;+\; N_{RG} \;+\; N_A \;+\; N_O \;+\; N_{V_{RG}} \tag{3.4}$$

Wie in Gleichung 3.4 zu erkennen ist, wird die Gesamtvariabilität durch Addition der verursachenden Elemente im betrachteten 150%-Modell *M1* bestimmt. Hierzu wird die Anzahl N der Alternativgruppen AG mit der Anzahl der Relationsgruppen RG sowie der Anzahl der Konfigurationsoptionen mit alternativer A, optionaler O und verpflichtender V_{RG} Variabilitätsbeziehung zusammengefasst. Dabei werden für $N_{V_{RG}}$ nur solche verpflichtenden Konfigurationsoptionen berücksichtigt, die einer Relationsgruppe zugeordnet sind und somit eine funktionale Beziehung zu anderen Konfigurationsoptionen in dieser Gruppe haben. Zudem werden für N_O sowohl unabhängige optionale Konfigurationsoptionen, als auch solche, die zu einer Relationsgruppe zugeordneten sind, berücksichtigt.

Die Ermittlung der Variabilität kann dabei entweder auf Ebene der Konfigurationsoptionen oder auf Ebene ihrer physischen Elemente erfolgen. Hierbei liegt es bei den Domänenexperten, welchen Fokus sie bei der Analyse und der späteren Restrukturierung legen wollen. Während unserer Arbeit mit den Experten unseres Industriepartners hat sich aber gezeigt, dass der Fokus stark auf den Konfigurationsoptionen liegt, da bei der Restrukturierung von TAPs beispielsweise die Ersetzung von logischen Komponenten eher von Interesse ist, als das Update solch einer Komponente von einer alten zu einer neueren Version. Daher fokussieren wir auch im Rahmen unserer Arbeit auf die Variabilität von Konfigurationsoptionen. Nichtsdestotrotz wird die Messung der Variabilität nachfolgend kurz für beide Ebenen gezeigt.

Für die Kandidaten unseres fortlaufenden Beispiels (vgl. Tabelle 3.14) ergibt sich durch die Anwendung der Gleichung 3.4 die in der folgenden Tabelle 3.15 ermittelte Gesamtvariabilität. Auf Ebene der physischen Elemente liegt die Variabilität in unserem Beispiel bei 21 und damit höher als die Variabilität auf Ebene der Konfigurationsoptionen (16). Dies ist der Fall, da hierfür jedes einzelne physische Element aller Konfigurationsoptionen berücksichtigt wird. So ist beispielsweise der Wert N_A=4 auf Ebene der Konfigurationsoptionen, da hier nur die vier alternativen Konfigurationsoptionen `A1-A4` gezählt werden. Auf Ebene der physischen Elemente

Ebene	$\mathbf{N_{AG}}$	$\mathbf{N_{RG}}$	$\mathbf{N_A}$	$\mathbf{N_O}$	$\mathbf{N_{V_{RG}}}$	**Gesamtvariabilität**
Konfigurationsoptionen	2	3	4	6	1	16
Physische Elemente	2	3	6	8	2	21

RG = Relationsgruppe *AG* = Alternativgruppe *V* = verpflichtend, *O* = optional, *A* = alternativ

Tabelle 3.15: Gemessene Variabilität für die identifizierten Kandidaten des fortlaufenden Beispiels

hingegen liegt dieser Wert bei N_A=6, da in `A1` und `A3` jeweils zwei verschiedene physische Elemente eingesetzt und somit einzeln berücksichtigt werden.

Die gemessene Variabilität eines generierten 150%-Modells gibt für sich betrachtet noch keinen Aufschluss darüber, ob diese unnötig oder zu hoch ist. Erst durch die Ableitung konkreter Restrukturierungsempfehlungen (vgl. Kapitel 4), können Kandidaten identifiziert werden, die tatsächlich zu unnötiger Variabilität führen und geeignete Maßnahmen zu deren Reduzierung bestimmt werden. Im Rahmen der *Differenzanalyse* (vgl. Unterabschnitt 5.1.1) lässt sich dann feststellen, wie hoch das Potential zur Reduzierung der gemessenen Variabilität für jede einzelne Restrukturierungsempfehlung ist. Zudem kann durch die späteren *Simulation* (vgl. Unterabschnitt 5.3.2) von ausgewählten Restrukturierungsempfehlungen die Variabilität des resultierenden Soll-150%-Modells bestimmt und mit der Variabilität des Ist-150%-Modells verglichen werden. Erst hierdurch lässt sich von Domänenexperten einschätzen, welcher Anteil der gemessenen Variabilität tatsächlich unnötig und damit zu hoch ist.

3.4 Verwandte Arbeiten

Im folgenden werden die verwandten Arbeiten beschrieben, die für dieses Kapitel der vorliegenden Dissertation von Bedeutung sind. Dazu wird zuerst auf Arbeiten eingegangen, die Variabilität im Kontext der EA-Domäne betrachten. Anschließend werden dann weitere relevante Arbeiten dargestellt, die Variabilität in der SPL-Domäne analysieren und bestimmen.

Bestimmung von Variabilität in der Enterprise Architecture Domäne

Im Kontext der EA-Domäne wird Variabilität aus zwei verschiedenen Perspektiven betrachtet. Zum einen findet die Analyse von Variabilität auf Ebene von *Elementen*, also einzelnen Bestandteilen einer Unternehmensarchitektur, statt. Zum anderen wird die Variabilität auch auf Ebene von *Modellen*, also zusammenhängenden Ausschnitten einer Unternehmensarchitektur, untersucht. Die verwandten Arbeiten für diese beiden Perspektiven werden nachfolgend beschrieben.

Variabilität von Elementen: In der EA-Domäne wird die Variabilität von Elementen vor allem im Rahmen des Forschungsfeldes der *IT-Komplexität* betrachtet. Hierbei werden häufig Begriffe wie *Heterogenität*, *Diversität* und *Varietät* anstelle des Terminus Variabilität verwendet. Diese Verschiedenartigkeit von Elementen wird als Teil der IT-Komplexität verstanden [SZM14].

Schneberger et al. [SM03] definieren IT-Komplexität mit dem Begriff *Computing Complexity*, der die Anzahl und die Varietät von Komponenten und ihren Beziehungen innerhalb einer IT-Landschaft sowie deren Änderungsrate betrachtet. In ihrer Arbeit führen sie das *Complexity Cross* ein, welches verdeutlicht, dass die Komplexität eines Softwaresystems von zwei gegenläufigen Tendenzen abhängt: die

abnehmende Komplexität von Komponenten versus die zunehmende Komplexität des Gesamtsystems. Wie die Varietät von Komponenten bestimmt werden kann, bleibt in dieser Arbeit allerdings offen.

Lagerström et al. [LBMA14] präsentieren einen Ansatz zur Bestimmung von abhängigen Elementen in einer IT-Architektur, um daraus zyklische Gruppen von Elementen abzuleiten und die Architekturen danach zu klassifizieren. Solche zyklischen Gruppen beinhalten Elemente, die entweder direkt oder indirekt von jedem anderen Element innerhalb der Gruppe abhängig sind. Dazu wird zuerst eine *Design Structure Matrix* erstellt, welche alle Beziehungen zwischen den verbauten Elementen einer Architektur abbildet. Aus dieser Matrix können dann alle Elemente den zyklischen Gruppen *Core*, *Control*, *Shared* und *Periphery* zugeordnet werden. Durch diese Gruppe werden versteckte Strukturen von Elementabhängigkeiten offen gelegt und können für weitere Analysen berücksichtigt werden.

Schütz et al. [SWK13] stellen den Begriff IT-Komplexität aus einer systemtheoretischen Perspektive dar. Hierbei wird die strukturelle Komplexität eines Softwaresystems durch die Anzahl und die Heterogenität der System-Elemente und deren Beziehungen bestimmt. Zur Messung der Heterogenität wird das *Entropie-Maß* nach Shannon [Sha48] vorgeschlagen. Dieses steigt mit der Anzahl unterschiedlicher Charakteristiken von Elementen sowie mit zunehmender Gleichverteilung. Proportionale Änderungen der Häufigkeiten wirken sich hingegen nicht auf das Entropie-Maß aus. Zwar ist hiermit ein erster Ansatz zur Messung von Heterogenität geliefert worden, allerdings fehlt die Möglichkeit der Operationalisierung für konkrete EA-Modelle.

Schmidt et al. [SWS13] greifen das Problem der fehlenden Operationalisierung von [SWK13] auf und entwickeln deren Ansatz weiter. Sie liefern eine mathematische Beschreibung zur Bestimmung der Heterogenität von Elementen mit Hilfe des Entropie-Maßes nach Shannon, erweitert um konkrete Artefakte der Unternehmensarchitektur. Hierzu stellen sie sowohl ein Metamodell als auch ein davon abgeleitetes Instanzmodell für Unternehmensarchitekturen vor, um konkrete Architekturelemente berücksichtigen zu können.

Widjaja et al. [WKTB12] führen mit ihrer Definition von Heterogenität eine andere quantifizierbare Beschreibung ein. Danach ist die Heterogenität in IT-Landschaften ein statistisches Maß, welches sich auf die Diversität von Elementattributen bezieht. Diese Definition ist bewusst so allgemein gewählt, dass Elemente aus verschiedenen Ebenen einer IT-Landschaft (z.B. aus der Geschäfts- oder Technologiearchitektur) betrachtet werden können. Dadurch können sie ein generisches Modell zur Bestimmung der Heterogenität entwickeln, welches zwei Anforderungen genügen muss. Zum einen soll die Heterogenität steigen, wenn die Anzahl der Werte für ein Elementattribut steigt (z.B. ein weiterer Hersteller) und zum anderen soll sie sich ebenfalls erhöhen, wenn die Disparität, also die Verschiedenheit, zwischen den einzelnen Werten eines Elementattributes sinkt.

Lakhrouit et al. [LB15] beschreiben ebenfalls einen Ansatz zur Messung der Komplexität von Komponenten einer Unternehmensarchitektur und verwenden hierfür ein Modell aus der Netzwerktheorie. Hierbei beschränken sie sich aber auf die Komponenten und deren Beziehungen und gehen explizit nicht auf die Varietät der

Komponenten ein. Dieses Konzept erweitern sie in [LB16], indem die Beschreibungssprache ArchiMate für die konkrete Modellierung von EA-Artefakten verwendet wird.

Über die eigentliche Berechnung der IT-Komplexität hinaus schlagen weitere Autoren konkrete Kennzahlen für die Ermittlung der IT-Komplexität sowie die darin enthaltene Heterogenität von Elementen vor.

Dern et al. [DJ09] stellen in ihrer Arbeit beispielsweise Maßgrößen zur Bestimmung der Komplexität von Applikationen vor, welche die Anzahl von Komponenten sowie deren Beziehungen und Homogenität berücksichtigen. Basierend auf Expertenschätzungen wurden diese Kennzahlen dann in [Der11] weiter konkretisiert.

Mocker [Moc09] untersuchte die Ursache-Wirkungs-Beziehungen der IT-Komplexität im Rahmen einer Fallstudie mit 273 Applikationen einer Investment Bank. Dabei hat er analysiert, wie sich die einzelnen Effekte der verschiedenen Komplexitätstypen *Interdependenz*, *Technologiediversität*, *Abweichung von Technologiestandards* sowie *Überschneidung/Redundanz* auf die Kosten für Betrieb und Wartung auswirken. Im Rahmen dieser Fallstudie hat er beispielsweise Kennzahlen wie *Anzahl der eingesetzten Betriebssysteme* oder *Anzahl der verwendeten Datenbanken* genutzt, um die Diversität zu messen.

Beetz et al. [BK11, Bee14] haben aufbauend auf der Arbeit von Mocker [Moc09] ein Modell zur Abbildung der IT Architektur Komplexität entwickelt, welches im Wesentlichen die Dimensionen enthält, wie sie auch von Mocker vorgeschlagen wurden. Zur Bestimmung der Diversität von Komponenten einer IT-Infrastruktur präsentieren sie die Kennzahlen *Anzahl der Software-Images für User/Server* und *Anzahl der Hardware Plattform Typen für User/Server*.

Schneider et al. [SM14, SRSM15] geben in ihrer Arbeit einen Überblick über Metriken, die dabei helfen, die Komplexität von Applikationslandschaften zu quantifizieren. Dabei zeigen sie zuerst Kennzahlen auf, die in der Literatur vorgeschlagenen werden und führen dann einen empirischen Vergleich mit Metriken durch, die sich in der Praxis zur Messung der Komplexität von Applikationslandschaften zu eignen scheinen. Hier zählen sie beispielsweise die *Anzahl und die Heterogenität von Komponenten eines bestimmten Typs (z.B. Applikationsserver)*, von *Datenbanken*, von *Betriebssystemen* und von *Schnittstellen* auf. Diese Metriken werden dann mit Branchenexperten auf ihre Anwendbarkeit hin untersucht und bewerten. Darüber hinaus haben die Autoren in [SSM16] die Heterogenität von Komponenten in Bezug auf ihre eingesetzten Versionen untersucht und eine quantitative Methode zur Bestimmung überalterter Komponenten in IT-Architekturen entwickelt. Hierfür verwenden sie ein *Schiefemaß*, welches klar herausstellt, ob eher veraltete oder neuere Versionen einer Komponente im Einsatz sind.

Vasconcelos et al. [VST07] stellen weitere 16 Metriken vor, die zur Bewertung von Softwaresystem-Architekturen verwendet werden können. Dabei wird mit den vorgeschlagenen Metriken das Ziel verfolgt, die Auswirkungen von Design-Entscheidungen für nicht-funktionale Qualitätsanforderungen im Vorfeld besser abschätzen zu können und so eine bessere Ausrichtung von Softwaresystemen zu Geschäftsanforderungen zu ermöglichen. Hierfür wird auch die Heterogenität von

Komponenten einer Applikation betrachtet. Diese kann beispielsweise mit Hilfe der *durchschnittlichen Anzahl von möglichen Betriebssystemen*, *Technologien* und *IT-Sicherheitskomponenten* bestimmt werden. Ebenfalls betrachten diese Metriken die Anzahl von Elementen und ihren Beziehungen.

Neben der Bereitstellung von Kennzahlen zur Messung von Heterogenität in IT-Architekturen, sind auch Frameworks zur Bestimmung von Diversität entwickelt worden. **Stirling** [Sti07] schlägt hierfür ein Framework vor, welches für den generischen Einsatz in verschiedenen Disziplinen, wie z.B. in der Technologie und der Gesellschaft, vorgesehen ist.

Schneider [Sch16] stellt aufbauend auf diesem generischen Ansatz ein konkretes Diversitäts-Framework für IT-Architekturen in seiner Arbeit vor. Hiermit ist es möglich, die Heterogenität von eingesetzten Komponenten über verschiedene Applikationen hinweg zu bestimmen. Schneider unterscheidet dabei zwischen Konzepten (logischen Elementen) und Individuen (physischen Elementen). Für die Individuen beschreibt er die Dimension *Variation*, welche die Unterschiede zwischen Attributen oder Merkmalen von physischen Elementen innerhalb eines Konzeptes untersucht. Außerdem stellt er die Dimensionen *Variety, Balance* und *Disparity* für Konzepte vor. Mit Variety werden die unterschiedlichen Konzepte beschrieben, die für einen spezifischen Anwendungszweck zum Einsatz kommen können. In der Dimension Balance wird analysiert, wie oft die unterschiedlichen Konzepte eingesetzt werden. Und im Rahmen der Dimension Disparity kann bestimmt werden, zu welchem Grad sich die einzelnen Konzepte unterscheiden.

Variabilität von Modellen: Im Kontext der Variabilität von Modellen werden in der EA-Domäne häufig Modelle von Unternehmensarchitekturen oder Teilen dieser, wie z.B. Modelle der Technologie- oder Geschäftsarchitektur, analysiert, um verschiedenartige Modelle mit einem Matching-Verfahren einander zuzuordnen oder zu integrieren sowie Abhängigkeiten und Unterschiede zu identifizieren.

Buckl et al. [BMS09] stellen in ihrer Arbeit einen Ansatz vor, unterschiedliche Beschreibungen des Enterprise Architecture Management (EAM) aus einer Viable System Model (VSM)-Perspektive zu analysieren und so unterschiedlichen EAM-Konzepte einem generischen VSM-Modell zuzuordnen. Hierfür verwenden sie die fünf Subsysteme *Enterprise-Level Management Process*, *Communication*, *Reactive EA Management*, *Proactive EA Management* und *EA Management Governance*.

Bakhshandeh et al. [BPB15, BPB16] stellen einen Ansatz vor, mit dessen Hilfe es möglich ist, EA-Modelle, welche in unterschiedlichen Modellierungssprachen beschrieben sind (z.B. ArchiMate und Business Process Model and Notation (BPMN)), auf ihre Gemeinsamkeiten und Unterschiede hin zu untersuchen. Hierfür analysieren sie die Heterogenität der Syntax, Struktur und Semantik solcher Modelle durch die Verwendung von Ontologie Matching Techniken. Um zu untersuchen, ob solche Techniken für das Matching von EA-Modellen geeignet sind, evaluieren sie ihr Verfahren mit *AgreementMakerLight*, einem Ontology Matching System.

Antunes et al. [ABCB15, ABM+14a, ABM+14b, ACB+13] stellen einen weiteren Ansatz vor, der Ontologien für die Analyse von EA-Modellen nutzt. Hierzu haben sie unterschiedliche EA-Modelle als ontologische Schemata spezifiziert. Anschließend wurden diese Schemata integriert und analysiert. Als Ergebnis haben sie in ihrer Arbeit ein vereinigtes Modell vorgestellt, welches durch eine Menge von ontologischen Schemata beschrieben wird. Für diese Beschreibung verwenden sie die Web Ontology Language (OWL) und integrieren unterschiedliche Modelle über zwei Kernelemente: die *Upper Ontologies* und die *Domain-Specific Ontologies*. Erstere bezeichnen sie als Schema oder Metamodell für die Repräsentation von domänenunabhängigen Kernentitäten und Beziehungstypen. Letztere stellt nach ihrer Ansicht eine DSL dar, welche ein wohl definiertes Set von Entitäten und Beziehungstypen für einen konkreten, begrenzten Kontext beschreibt. Durch die Integration solcher Domain-Specific Ontologies in Upper Ontologies entsteht so eine Gesamtbeschreibung für ein integriertes EA-Modell.

Sunkle et al. [SKR13] präsentieren ebenfalls einen Ansatz zur Analyse von EA-Modellen mit Hilfe von Ontologien. Sie zeigen in ihrer Arbeit, wie bestehende Modellierungssprachen für EA-Modelle am Beispiel von ArchiMate verwendet werden können, um eine EA-Ontologie zu entwickeln. Darüber hinaus stellen sie auch dar, wie auf Basis solch einer Ontologie eine Analyse der EA-Modelle erfolgen kann. So skizzieren sie beispielsweise die *Landscape Mapping Analyse*, mit deren Hilfe Abhängigkeiten (z.B. associatedWith, usedBy, assignedTo) zwischen unterschiedlichen Elementen einer Unternehmensarchitektur identifiziert werden können.

Buschle et al. [BJS13, NBE14] stellen in ihrer Arbeit ein Framework zur Bewertung von Merkmalen verschiedenster EA-Modelle vor, welches auf Basis des ArchiMate-Metamodells entwickelt wurde. Dieses Framework integriert vier existierende Metamodelle zu einem, um die Eigenschaften *Anwendungsnutzung*, *Systemverfügbarkeit*, *Service-Reaktionszeit* und *Genauigkeit der Daten* zu untersuchen. Hierfür werden die unterschiedlichen Metamodelle zu einem Gesamt-Metamodell integriert und schließlich vier verschiedene *Viewpoints* zur Analyse der konkreten Merkmale herausgearbeitet.

Roth et al. [RHZ+13, HRPM13] präsentieren einen Matching-Ansatz für EA-Modelle mit dem Ziel, nicht-technische Anwender bei der Analyse unterschiedlicher EA-Modellen durch eine dynamische und konfigurierbare Visualisierung zu unterstützen. Hierfür stellen sie ein Pattern-Matching Algorithmus vor, mit deren Hilfe strukturelle EA-Modelle analysiert und die benötigten Informationen auf Basis der erstellten Konfiguration an die Visualisierung gebunden werden können. Für dieses Verfahren stellen sie ebenfalls ein Tool vor, welches bei der Konfiguration und Erstellung solcher Visualisierungen unterstützt.

Die vorgestellten Ansätze zur Bestimmung der Variabilität von Elementen und Modellen aus der EA-Domäne sind gut geeignet, um die Heterogenität im Hinblick auf spezifische Problemstellungen zu analysieren. Allerdings bieten sie nicht die Möglichkeit, konkrete Variabilitätsbeziehungen zwischen einzelnen Elementen einer Technologiearchitektur zu bestimmen (zB. verpflichtende, optionale und alternative Bezie-

hungen). Mit Hilfe unseres Variabilitätsmining-Verfahrens können Domänenexperten solche detaillierten Variabilitätsinformationen nun ohne hohen manuellen Aufwand für eine große Anzahl an Softwaresysteme identifizieren und auf dieser Basis weitere Analysen durchführen sowie geeignete Architekturentscheidungen treffen.

Bestimmung von Variabilität in der Software Product Line Domäne

Einen guten Überblick über verfügbare Ansätze zum Management von Variabilität in Softwaresystemen geben **Babar et al.** [BCS10] und **Galster et al.** [GWT+14]. Darüber hinaus existieren viele verschiedenen Ansätze in der Literatur, mit deren Hilfe die Variabilität zwischen unterschiedlichen Implementierungsartefakten (u.a. Source-Code (z.B. [LLHE17]) oder Modelle (z.B. [Wil18])) identifiziert werden kann. Im Kontext dieser Arbeit sind vor allem solche Ansätze als verwandte Arbeiten zu berücksichtigen, welche die Variabilität von Modellen mit Hilfe geeigneter Mining-Verfahren bestimmen können. Solche Variabilitätsmining-Verfahren lassen sich dabei zwischen *paarweisen* und *n-way* Algorithmen unterscheiden.

Verfahren mit paarweisen Algorithmen: Die folgenden Arbeiten stellen paarweise Verfahren vor, die sich dadurch auszeichnen, dass sie jeweils zwei Artefakte gleichzeitig miteinander vergleichen.

Font et al. [FAHC15, FBHC15] präsentieren einen Ansatz mit dem sie verwandte Modellvarianten in eine SPL, die auf einer CVL basiert, überführen. Hierfür bestimmen sie zunächst die Differenzen für alle paarweisen Kombinationen der betrachteten Modellvarianten, bevor sie auf Grundlage dieser errechneten Differenzen das Modell, welches die geringsten Unterschiede zu anderen Modellen aufweist, als Basismodell auswählen. Unter Verwendung der identifizierten Differenzen können dann Variationspunkte definiert und mögliche Varianten von eingesetzten Artefakten zugeordnet werden. Zudem bietet der Ansatz eine geeignete Unterstützung an, um aus dem Basismodell alle betrachteten Modellvarianten ableiten zu können.

Nejati et al. [NSC+07, NSC+12] stellen in ihrer Arbeit ein Verfahren zur Manipulation von Merkmals-Spezifikationen für hierarchische Modelle von Zustandsautomaten vor. Hierzu nutzen sie die beiden Operatoren *Match* und *Merge*. Der erste Operator dient dazu, Übereinstimmungen zwischen zwei Modellen zu identifizieren. Mit dem zweiten Operator lassen sich anschließend Modelle, die bekannte oder angenommene Übereinstimmungen aufweisen, integrieren. Der Match-Operator ist heuristisch und verwendet sowohl statische als auch dynamische Merkmale der Modelle. Der Merge-Operator hingegen hält die hierarchische Struktur der Eingabemodelle vor und verwendet Parametrisierung zur Behandlung unterschiedlicher Verhaltensmerkmale der Modelle. Dadurch ist eine automatisch Zusammenführung der Modelle unter Beibehaltung der Semantik der hierarchischen Zustandsautomaten möglich.

Rubin et al. [RC12, RC13b] schlagen ebenfalls einen Ansatz zur Analyse von Zustandsautomaten vor, welcher einen *Merge-In-Operator* verwendet. Mit Hilfe dieses Operators werden verwandte Software-Varianten paarweise und iterativ miteinander

verglichen, wodurch Gemeinsamkeiten zwischen ihnen auf Grundlage von Ähnlichkeitswerten bestimmt werden können. Auf Basis dieser Ergebnisse können dann die einzelnen, unterschiedlichen Teile der analysierten Modelle zu einem Gesamtmodell integriert werden. Durch die Verwendung von Annotationen wird sichergestellt, dass die Information, welcher Teil zu welcher Variante gehört, auch im Gesamtmodell vorhanden ist.

Wille et al. [WSS16, WSSS16, Wil18] präsentieren in ihrer Arbeit ein Verfahren, um die Variabilität von block-basierten Modellierungssprachen zu identifizieren. Ihr Algorithmus besteht aus den drei Phasen *Compare*, *Match* und *Merge* und stellt einen generischen Ansatz dar, der für beliebige block-basierte Modellierungssprachen verwendet werden kann. In der Compare-Phase werden einzelne Modellelemente paarweise miteinander verglichen und ihre Übereinstimmung mittels Ähnlichkeitswerten bestimmt. In der Match-Phase werden für Mehrdeutigkeiten, die in der ersten Phase identifiziert wurden, konkrete Beziehung zwischen exakt zwei Modellen hergestellt. Basierend darauf wird in der Merge-Phase ein 150%-Modell erstellt, welches alle Modellelemente der verglichenen Varianten zusammenfügt und deren entsprechende Variabilität annotiert.

Verfahren mit n-way Algorithmen: Neben den paarweisen Verfahren gibt es auch verschiedene n-way Verfahren zur Bestimmung von Variabilität. Sie zeichnen sich durch parallele Vergleiche von beliebig vielen Artefakten aus. Relevante Arbeiten in diesem Kontext werden nachfolgend vorgestellt.

Rubin et al. [RC13a] entwickeln ihren paarweisen Ansatz (vgl. [RC12]) weiter, da sie feststellen, dass ein Vergleich von nur zwei Artefakten zur gleichen Zeit zu Ergebnisse führen kann, die insgesamt nicht dem Optimum über alle Artefakte entsprechen. So kann beispielsweise ein Merging von zwei Elementen in einer frühen Iteration des Verfahrens verhindern, dass ein sinnvolleres Merging mit einem anderen Element zu einer späteren Iteration erfolgen kann, wodurch unerwartete Variabilitätsbeziehungen erkannt werden könnten. Zur Lösung dieses Problems stellen die Autoren einen heuristischen Ansatz vor, der durch einen n-way Algorithmus das Optimum für alle zu vergleichenden Elemente findet. Mit diesem Algorithmus werden Tupel von Modellelementen gebildet und durch Verkettung aneinandergereiht. Hierdurch kann die höchstmögliche Ähnlichkeit über alle Modellelementen bestimmt werden.

Martinez et al. [MZB+15, MZKT14] stellen in ihrer Arbeit ein Verfahren zur Analyse von Modellvarianten vor, welches sie als *Model Variants Comparison (MoVaC)* bezeichnen. Mit Hilfe dieses Ansatzes können beliebig viele Modellvarianten miteinander verglichen werden, um so deren Gemeinsamkeiten und Unterschiede in Form von *Features* zu identifizieren. Solche Features stellen Sets von unabhängigen, atomaren Modellelementen dar. Mit MoVaC kann die Variabilität von Modellen auf Basis dieser atomaren Modellelemente identifiziert werden und durch eine geeignete grafische Repräsentation dem Benutzer explizit visualisiert werden.

Zhang et al. [ZHMP11, ZHMP12] schlagen einen weiteren Ansatz für Modellvergleiche vor. Mit Hilfe eines generischen Verfahrens werden Unterschiede in betrach-

teten Softwaremodellen identifiziert, um daraus eine Software Produkt Linie abzuleiten. Dazu werden Meta-Object Facility (MOF)-basierte Modelle von potentiellen Produktvarianten mit einem vom Entwickler ausgewählten Basismodell verglichen. Durch *low-level* und *high-level* Vergleiche können Gemeinsamkeiten und Unterschiede sowohl zwischen den einzelnen Modellen als auch für die gesamte Domäne bestimmt werden. Mit den Ergebnissen der Vergleiche kann dann ein CVL-basiertes Feature Modell erstellt werden, welches die Variationspunkte und Varianten (auch als *Modellfragmente* bezeichnet) der Modelle darstellt. Dieses Feature Modell kann dann vom Entwickler manuell auf Basis seines Domänenwissens verbessert und somit an die jeweiligen Bedürfnisse angepasst werden.

Shatnawi et al. [SSS15] präsentieren einen Reverse-Engineering-Ansatz zur Identifikation der Variabilität von Elementvarianten einer Software-Architektur und deren Abhängigkeiten. Hierzu bestimmen sie Varianten von Software-Komponenten anhand der Ähnlichkeit ihrer Klassennamen. Abhängigkeiten zwischen solchen Komponenten bestimmen sie auf Basis von deren Einsatz für die Produkt Architektur. Der vorgeschlagene Ansatz basiert dabei auf der Formal Concept Analysis (FCA) und wurde mit zwei verschiedenen Open-Source Produktvarianten evaluiert.

Acher et al. [ACM+11] stellen einen weiteren Reverse-Engineering-Ansatz vor, um die Variabilität von Softwareartefakten zu identifizieren. Hierfür präsentieren sie in ihrer Arbeit einen umfangreichen und Tool-unterstützen Prozess, mit dem Feature Modelle für Softwaresysteme erstellt werden können. Durch die Anwendung automatisierter Techniken für die *Extraktion*, die *Aggregation*, das *Alignment* und die *Projektion* von Softwarearchitekturen können konkrete Feature Modelle für die analysierten Softwaresysteme generiert werden, welche deren Variabilität beschreiben.

Die vorgestellten Verfahren unterscheiden sich in der Art und Weise, wie sie die identifizierte Variabilität darstellen. So nutzen einige Ansätze die CVL zur Beschreibung von Variabilität (z.B. [FAHC15, FBHC15, MZB+15, ZHMP11]), wohingegen andere Ansätze 150%-Modelle für diesen Zweck verwenden (z.B. [NSC+07, RC12, RC13a, SSS15, WSSS16]). Obwohl all diese Verfahren in der Lage sind, Variabilität in Modellen von Softwaresystemen zu bestimmen, können die meisten von ihnen keine konkreten Variabilitätsbeziehungen zwischen Elementen identifizieren (also beispielsweise verpflichtende, optionale oder alternative Beziehungen). Oft werden lediglich die identifizierten Varianten einem analysiertem Element zugeordnet. Für die Bestimmung von Variabilität in Technologiearchitekturen gäbe es zwar theoretisch Ansätze, die generisch genug sind, um sie für die Analyse von TAP-Modellen zu verwenden (z.B. [ZHMP11, MZB+15]), jedoch berücksichtigen diese keine strukturellen Besonderheiten (z.B. Layer-Tier-Strukturen oder Kategoriensysteme von Elementen). Somit erlauben die existierenden Ansätze aus der SPL-Domäne kein effizientes Variabilitätsmining für Technologiearchitekturen.

3.5 Zusammenfassung

In diesem Kapitel haben wir ein Verfahren zur Bestimmung der Variabilität von Technologiearchitekturen für gewachsene IT-Landschaften vorgestellt. Mit Hilfe dieses Ansatzes können Domänenexperten automatisiert eine beliebige Anzahl von TAPs analysieren und die gemeinsamen sowie variierenden technologischen Bestandteile der betrachteten Softwaresysteme identifizieren. Dadurch ist es möglich, explizite Variabilitätsbeziehungen zwischen den einzelnen verbauten Komponenten zu bestimmen und so verpflichtende, alternative, optionale und funktional verwandte Elemente zu identifizieren.

Hierfür wurde zuerst in Abschnitt 3.1 gezeigt, welche vorbereitenden Maßnahmen für die Analyse der zur Verfügung stehenden Daten über Softwaresysteme erforderlich sind. So sollten TAPs beispielsweise auf Basis eines gleichen Verwendungszweckes (z.B. SAP-Applikationen) in spezifische Sets zusammengeführt und anschließend in das interne Datenformat des vorgestellten Verfahrens überführt werden, um eine effiziente Verarbeitung von artverwandten Softwaresystemen zu ermöglichen. Darüber hinaus wurden Techniken wie z.B. die Strukturanalyse und die Ausreißersuche vorgestellt, welche die Qualität der vorhandenen Daten vor der Analyse (semi-)automatisiert verbessern können.

In Abschnitt 3.2 ist dann das Variabilitätsmining-Verfahren für Technologiearchitekturen beschrieben worden. Dieses ist in der Lage, eine beliebige Anzahl von TAPs gleichzeitig zu analysieren und alle Gemeinsamkeiten und Unterschiede zu identifizieren sowie in einem einzigen 150%-Modell abzubilden. Hierfür werden zuerst alle technologischen Komponenten der betrachteten TAPs, die in einer spezifischen Intersection verbaut sind, in einem Cluster gruppiert. Danach werden unabhängige Konfigurationsoptionen in solch einem Cluster durch Splitting-Regeln extrahiert. Anschließend werden die konkreten Variabilitätsbeziehungen zwischen den Komponenten eines Clusters bestimmt und diese Komponenten dann mit ihren jeweiligen Variabilitätsinformationen in ein gemeinsames 150%-Modell zusammengeführt. Durch diese 150%-Modell-Ansicht wird ein zusätzlicher Mehrwert für Domänenexperten geschaffen, da Variabilitätsbeziehungen zwischen beteiligten Komponenten und damit Korrelationen (z.B. alternative Komponenten) ohne großen manuellen Aufwand erkannt werden können.

Im letzten Abschnitt 3.3 wurde gezeigt, wie Kandidaten bestimmt werden, die zu unnötiger Variabilität führen können. Hierzu ist zuerst beschrieben worden, wie funktionale Beziehungen zwischen einzelnen Komponenten ermittelt werden können, um auf Grundlage dieser Ergebnisse und des 150%-Modells konkrete Kandidaten für unnötige Variabilität identifizieren zu können. Abschließend ist dann dargestellt worden, wie die Variabilität, die durch solche Kandidaten verursacht wird, gemessen werden kann. Diese Ergebnisse zeigen Domänenexperten Potentiale für Wiederverwendungen von Komponenten und für mögliche Restrukturierungen von TAPs auf. Zudem stellen sie den Input für das Verfahren des nachfolgenden Kapitel 4 dar, welches Domänenexperten bei der Analyse von Restrukturierungspotentialen durch Ableitung konkreter Empfehlungen unterstützt.

4 Restrukturierungsempfehlungen zur Reduzierung unnötiger Variabilität

Dieses Kapitel basiert in großen Teilen auf den Veröffentlichungen [WWSS17a] und [WS17].

Die Identifizierung und Reduzierung von unnötiger Variabilität in technischen Architekturen ist eine herausfordernde und zeitraubende Tätigkeit für Domänenexperten, da hierfür keine automatisierten Techniken zur Verfügung stehen und somit hoher manueller Aufwand erforderlich ist. Daher haben wir einen neuen Ansatz entwickelt, der eine automatische Ableitung von konkreten *Restrukturierungsempfehlungen (RE)* für eine beliebige Menge von TAPs ermöglicht und so Experten bei der Reduzierung unnötiger Variabilität unterstützt. Wie in Abbildung 4.1 dargestellt, besteht unser Ansatz dabei aus drei Phasen und baut auf den Ergebnissen des letzten Kapitels (Kapitel 3) auf. Dabei wird mit dem hier vorgestellten Ansatz die *Phase D* unseres Gesamtverfahrens (vgl. Abbildung 1.1) umgesetzt.

In der ersten Phase *Regelbasierte Businessanalyse* (vgl. Abschnitt 4.1) werden dafür die aus dem 150%-Modell abgeleiteten Kandidaten für unnötige Variabilität (vgl. Definition 3.8) aus einer businessorientierten Perspektive analysiert. Im Fokus steht dabei eine auf Geschäftsregeln basierte Analyse, welche von Experten definierte Geschäftsanforderungen und -ziele zur Bewertung der Kandidaten verwendet, um dar-

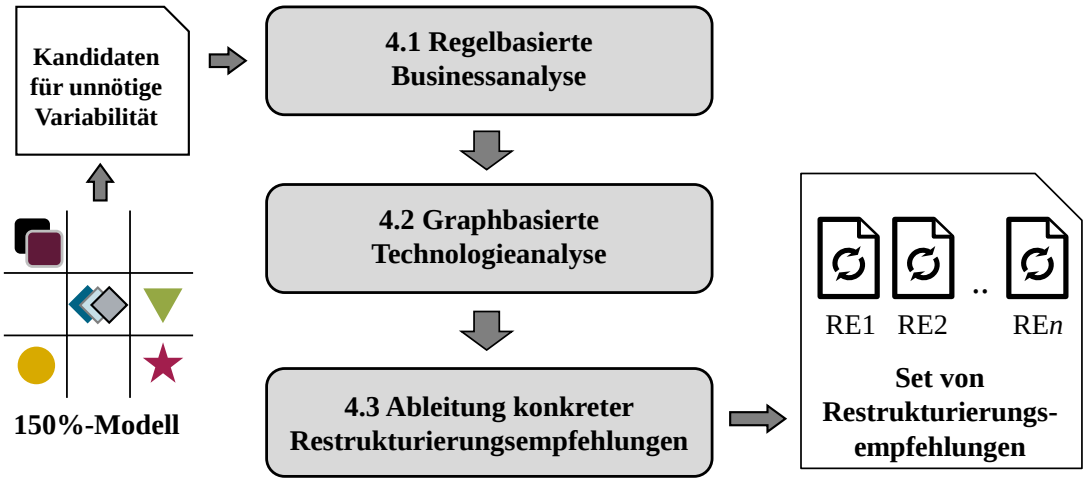


Abbildung 4.1: Workflow für die Ableitung von Restrukturierungsempfehlungen

aus Restrukturierungspotentiale abzuleiten. Diese Restrukturierungspotentiale werden anschließend in der *Graphbasierten Technologieanalyse* (vgl. Abschnitt 4.2) aus einer technischen Sicht analysiert und bewertet. Hierfür werden die betroffenen TAPs einzelner Kandidaten als Graphen dargestellt und miteinander verglichen, um die technische Machbarkeit von Restrukturierungspotentialen bestimmen zu können. Die Ergebnisse beider Phasen gehen dann in die dritte Phase *Ableitung konkreter Restrukturierungsempfehlungen* (vgl. Abschnitt 4.3) ein. Hier wird untersucht, ob ein Kandidat sowohl aus businessorientierter als auch aus technischer Sicht als nicht erforderlich angesehen werden kann und damit tatsächlich für unnötige Variabilität sorgt. In solch einem Fall wird eine konkrete Restrukturierungsempfehlung abgeleitet, um den betrachteten Kandidaten eliminieren zu können.

4.1 Analyse von Kandidaten aus businessorientierter Sicht

In vielen Unternehmen werden IT-Ziele von allgemeinen Geschäftszielen abgeleitet, woraus sich Anforderungen an die gesamte Technologiearchitektur ergeben (vgl. Abschnitt 2.2). Diese Anforderungen und Ziele spielen eine wesentliche Rolle bei der Analyse und Bewertung von Kandidaten für unnötige Variabilität, da sie die Grundlage für Entscheidungen von Domänenexperten im Hinblick auf die Rationalisierung von technologischen Komponenten einer IT-Architektur bilden.

Um solche Geschäftsanforderungen und -ziele berücksichtigen zu können, stellen wir in diesem Abschnitt die *Regelbasierte Businessanalyse* (vgl. Abbildung 4.2) vor,

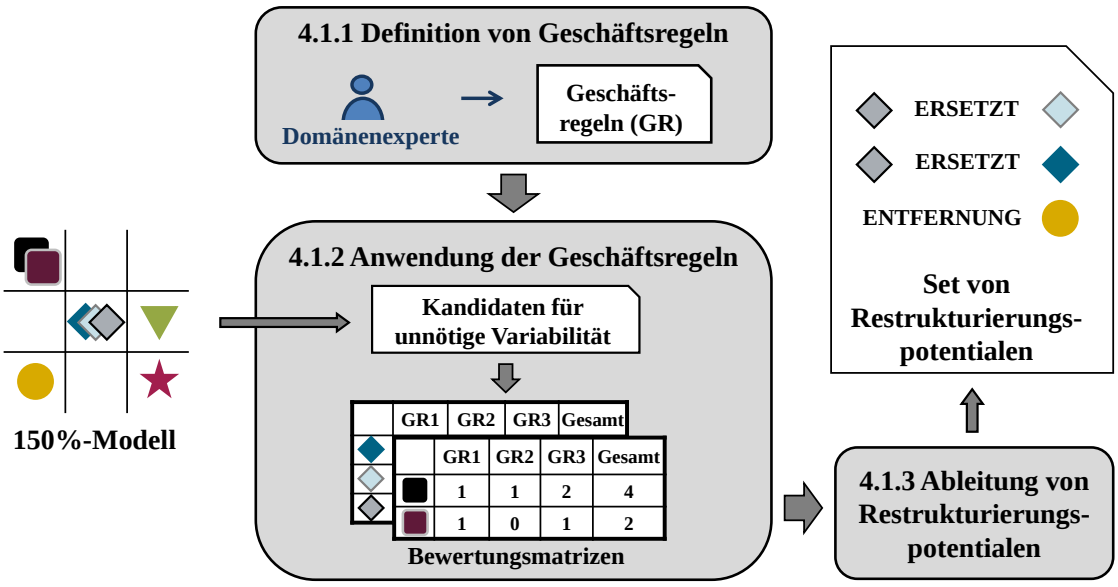


Abbildung 4.2: Workflow für die regelbasierte Businessanalyse

welche die Entscheidungsfindung von Domänenexperten auf Basis von formalisierten Geschäftsregeln nachbildet und somit eine automatisierte Bewertung von Kandidaten für unnötige Variabilität aus einer businessorientierten Sicht erlaubt.

Wie Abbildung 4.2 zeigt, müssen Domänenexperten zunächst ihre Anforderungen und Ziele an die Technologiearchitektur mit Hilfe von *Geschäftsregeln (GR)* definieren (vgl. Unterabschnitt 4.1.1). Anschließend können diese Regeln einzeln auf die identifizierten Kandidaten für unnötige Variabilität angewendet und die resultierenden Ergebnisse in einer Bewertungsmatrix zusammengefasst werden (vgl. Unterabschnitt 4.1.2). Auf Basis dieser Matrix können schließlich Restrukturierungspotentiale identifiziert werden, die solche Kandidaten, welche aus einer businessorientierten Sicht als *nicht erforderlich* gelten, ersetzen oder entfernen (vgl. Unterabschnitt 4.1.3).

4.1.1 Geschäftsregeln zur Abbildung fachlicher Anforderungen

Um Kandidaten für unnötige Variabilität automatisiert aus einer businessorientierten Perspektive analysieren und bewerten zu können, ist es erforderlich, dass Domänenexperten Anforderungen und Ziele an die Technologiearchitektur in Form von Geschäftsregeln (GR) beschreiben.

Definition 4.1: Geschäftsregel

Im Allgemeinen wird unter dem Begriff *Geschäftsregel (GR)* ein Aussage verstanden, die eine bestimmten Aspekt einer Unternehmung steuert. Dabei kann dieser Begriff sowohl aus einer Unternehmens-, als auch aus einer Systemperspektive betrachtet werden [Ros03b].

Im Laufe der Zeit haben sich verschiedene Arten von Geschäftsregeln entwickelt. Zu den wichtigsten zählen dabei die folgenden drei [TW01]:

Einschränkungsregeln: Beschreiben Aussagen, die immer wahr sind und somit uneingeschränkt gelten. Sie können als Gebot oder Verbot definiert werden.
Beispiel: Jeder Kunde erhält eine Kundennummer

Ableitungsregeln: Beschreiben Aussagen, die neues Wissen durch Schlussfolgerung oder mathematische Bestimmung ableiten.
Beispiel: Ein Kunde erhält eine Mahnung, falls seine letzte Rechnung mehr als 14 Tage überfällig ist.

Reaktionsregeln: Beschreiben Aussagen, die bestimmte Aktionen als Antwort auf ein Ereignis anstoßen.
Beispiel: Wenn sich ein Kunde einloggt, dann wird ihm eine Übersicht seiner letzten Bestellungen angezeigt.

Im Kontext dieser Arbeit werden *Ableitungsregeln* für die Businessanalyse benötigt. Diese erlauben eine Beschreibung der Geschäftsanforderungen und -ziele von Domänenexperten mit Hilfe von Aussagen, die eine Schlussfolgerung hinsichtlich der Notwendigkeit eines identifizierten Kandidaten für unnötige Variabilität ermöglichen. Im Folgenden sind drei Beispiele für solche Ableitungsregeln dargestellt:

GR1: *Wenn ein Kandidat einer Variabilitätsgruppe eine Nutzungshäufigkeit von mehr als 66% hat, dann ist er erforderlich, mit einem Gewicht von 3.*

GR2: *Wenn ein Kandidat eine Nutzungshäufigkeit von weniger als 10% hat und eine unabhängige optionale Konfigurationsoption darstellt, dann ist er nicht erforderlich.*

GR3: *Wenn ein Kandidat die geringsten Total Cost of Ownership (TCO)*[1] *innerhalb einer Variabilitätsgruppe aufweist, dann ist er erforderlich.*

Um solche Geschäftsregeln im Rahmen der Businessanalyse automatisiert verarbeiten zu können, müssen diese in einer formalen Beschreibung vorliegen. Hierfür wird ein verständlicher und benutzerfreundlicher Ansatz benötigt, der es Domänenexperten ermöglicht, ihre Anforderungen und Ziele mit geringem manuellen Aufwand durch Verwendung von Begrifflichkeiten aus der EA-Domäne zu beschreiben. Zwar bieten einige Ansätze, wie beispielsweise die *Event Condition Action (ECA) Regel* [Day95] oder die *Ereignisgesteuerte Prozesskette (EPK)* [Sch02], bereits gute Möglichkeiten zur formalen Darstellung von Geschäftsregeln, jedoch steht hierbei ein eintretendes Ereignis im Vordergrund, weshalb sich diese Ansätze eher zur Beschreibung von Reaktionsregeln eignen. Eine weitere Möglichkeit zur Spezifizierung von Regeln bietet das *Behavior Driven Development (BDD)*, welches nicht nur auf Ereignisse fokussiert ist, sondern auch das Verhalten eines Systems berücksichtigt.

Definition 4.2: Behaviour Driven Development

Das Behavior Driven Development (BDD) ist ein Ansatz zur Entwicklung von Anwendungssystemen, bei dem die Spezifikation des Verhaltens vom zu entwickelnden System im Fokus steht. Dabei wird mit BDD das Ziel verfolgt, ausführbare Spezifikationen des Systems zu entwickeln, sodass sie automatisiert verarbeitet werden können [SW11].

Im Vordergrund von BDD steht dabei die Spezifizierung von *Szenarien*, welche das Verhalten eines Systems beschreiben. Solch ein Szenario kann für einen spezifischen Kontext definieren, wie sich der Zustand des betreffenden Softwaresystems verändern soll, wenn ein spezifisches Ereignis eintritt.

[1] *TCO* beinhaltet verschiedene kostenrelevante Aspekte einer Komponente, z.B. die Kosten für den Erwerb, die Lizenzierung und die Wartung der betrachteten Komponente sowie für die Qualifizierung der Mitarbeiter [Ell94]

Die allgemeine Struktur zur Beschreibung eines solchen Szenarios besteht aus den folgenden drei Bestandteilen [SW11]:

1. *Szenario*: [Szenario Titel]
2. *GIVEN*: [Kontext]
3. *WHEN*: [Event]
4. *THEN*: [Ergebnis]

Mit dem Präfix `GIVEN` wird der Kontext beziehungsweise der Zustand beschrieben, in welchem sich das betrachtete System zu Beginn des Szenarios befinden muss. Ein eintretendes Ereignis kann mit dem Präfix `WHEN` beschrieben und das resultierende Ergebnis mit dem Präfix `THEN` definiert werden. Zur Erstellung der Beschreibung in den eckigen Klammern wird das Konzept der *Ubiquitous Language* verwendet. Es beschreibt eine Sprache, welche von einem Domänenmodell abgeleitet ist und die Terminologie dieser Domäne für die Beschreibung des Verhaltens eines Systems nutzt [SW11].

Ein beispielhaftes Szenario für die Geldabhebung an einem Bankautomaten könnte wie folgt aussehen:

1. *Szenario*: Bargeldauszahlung
2. *GIVEN*: EC-Karte ist im Bankautomat eingesteckt und PIN ist verifiziert
3. *WHEN*: Geldbetrag ist ausgewählt
4. *THEN*: Überprüfung des Geldrahmens und Auszahlung des gewählten Geldbetrages

Dieses Beispielszenario greift nur dann, wenn der Bankautomat sich in dem beschrieben Zustand befindet (`GIVEN`). Bei Eintreten des Ereignisses *Geldbetrag ist ausgewählt* (`WHEN`) erfolgt die Überprüfung und Auszahlung des Geldbetrages (`THEN`).

Zur Spezifizierung der erforderlichen Ableitungsregeln für die Businessanalyse setzen wir auf dem Konzept des BDDs auf und schaffen damit die Möglichkeit, solche Geschäftsregeln mit Hilfe von Szenarien zu beschreiben. Um dabei den manuellen Aufwand für Domänenexperten so gering wie möglich zu halten, haben wir eine *Domain-Specific Language (DSL)* entworfen, welche konkrete Notationen für die spezifische Problemdomäne von EA-Experten bietet und sich dabei auf die relevanten Aspekte dieser Domäne fokussiert. Die entwickelte DSL wird nachfolgend näher erläutert.

Domain Specific Language zur Formalisierung von Geschäftsregeln

Mit Hilfe unserer DSL ist es möglich, Geschäftsanforderungen und -ziele in Form von *Geschäftsregeln* formal zu beschreiben. Zur Spezifizierung einer solchen Geschäftsregel `GRi` haben wir zusammen mit Domänenexperten aus unserem Industrieunternehmen eine konkrete Syntax für unsere DSL entworfen, welche die nachfolgende Struktur aufweist. Diese besteht aus fünf Teilen, die jeweils durch einen spezifischen Präfix eingeführt werden:

1. *`RULE`*: [Titel der Geschäftsregel]
2. *`GIVEN`*: [Voraussetzung]
3. *`WHEN`*: [Bedingung]
4. *`THEN`*: [Konsequenz]
5. *`WEIGHT`*: [Gewichtungsfaktor]

Durch den Präfix `RULE` wird ein beschreibender Titel für eine Geschäftsregel bestimmt. Mit Hilfe des Präfix `GIVEN` können die Voraussetzungen definiert werden, die erfüllt sein müssen, damit eine Geschäftsregel auf konkrete Kandidaten, also auf Konfigurationsoptionen des betrachteten 150%-Modells (vgl. Definition 3.8), angewendet werden kann. Hierzu zählen beispielsweise, ob und zu welcher Variabilitätsgruppe solch eine Konfigurationsoption gehören muss oder welche Daten und Informationen (z.B. Daten über Kosten für Komponenten) zur Verfügung stehen müssen. Erst wenn diese Voraussetzungen erfüllt sind, wird die Geschäftsregel auch auf einen Kandidaten angewandt. Im `WHEN`-Teil kann dann die konkrete Geschäftsanforderung beziehungsweise das konkrete Geschäftsziel beschrieben werden. Hierbei handelt es sich häufig um Schwellwerte für ein bestimmtes Charaktermerkmal. Beispielsweise kann definiert werden, dass ein Kandidat in mindestens 60% aller Fälle eingesetzt sein soll, um für eine Standardisierung berücksichtigt zu werden. Solche Anforderungen stellen die Bedingung zur Ausführung der Geschäftsregel dar. Erst wenn alle Bedingungen einer Geschäftsregel erfüllt sind, wird auch die Konsequenz als resultierendes Ergebnis der jeweiligen Regel umgesetzt. Diese Konsequenz ist mit dem Präfix `THEN` beschrieben und bestimmt, ob ein Kandidat bei Erfüllung der Geschäftsregel als *erforderlich* oder als *nicht erforderlich* betrachtet wird. Um die Relevanz einzelner Geschäftsregeln hervorzuheben und sie gegenüber anderen Regeln mit einem größeren Gewicht zu versehen, kann mit dem `WEIGHT`-Präfix ein Gewichtungsfaktor eingeführt werden, der den Kandidaten, welche diese Regel erfüllen, ein größeres Gewicht verleiht.

Mit Hilfe dieser Beschreibung kann beispielsweise die Geschäftsregel `GR1` (*Wenn ein Kandidat einer Variabilitätsgruppe eine Nutzungshäufigkeit von mehr als 66% hat, dann ist er erforderlich, mit einem Gewicht von 3*) wie folgt definiert werden.

1. *RULE*: GR1 – Standardize candidates in variability groups with usage frequency greater than 66%
2. *GIVEN*: Information on variability and frequencies is available, candidate is part of related group or alternative group
3. *WHEN*: candidate has usage frequency greater than 66%
4. *THEN*: candidate is required
5. *WEIGHT*: 3

Zudem kann die Geschäftsregel `GR2` (*Wenn ein Kandidat eine Nutzungshäufigkeit von weniger als 10% hat und eine unabhängige optionale Konfigurationsoption darstellt, dann ist er nicht erforderlich*) wie folgt beschrieben werden.

1. *RULE*: GR2 – Remove independent optional candidates with usage frequency less than 10%
2. *GIVEN*: Information on variability and frequencies is available, candidate is independent and optional
3. *WHEN*: candidate has usage frequency less than 10%
4. *THEN*: candidate is not required

Außerdem lässt sich die Geschäftsregel `GR3` (*Wenn ein Kandidat die geringsten TCO innerhalb einer Variabilitätsgruppe aufweist, dann ist er erforderlich*) wie folgt beschreiben.

1. *RULE*: GR3 – Standardize candidates in variability groups with lowest TCO
2. *GIVEN*: Information on variability and TCO is available, candidate is part of related group or alternative group
3. *WHEN*: candidate has lowest TCO
4. *THEN*: candidate is required

Um den manuellen Aufwand von solchen formalen Beschreibungen zu reduzieren, haben wir eine DSL entwickelt, welche die Spezifizierung von Geschäftsregeln ermöglicht und dadurch deren automatisierte Verarbeitung erlaubt. In den nachfolgenden Listings 4.1 bis 4.4 stellen wir die Grammatik, also die konkrete Syntax, für die entworfene DSL anhand von Auszügen vor. Diese wurde mit Hilfe der vorgestellten Xtext-Notation entwickelt (vgl. Abschnitt 2.3). Die vollständige Grammatik für diese DSL kann im Anhang (vgl. Abschnitt A.1) eingesehen werden.

```
Rule:
   'RULE' id = STRING ('description' title = STRING)?
       givenRule = GivenRulePart
       whenRule = WhenRulePart
       thenRule = ThenRulePart
       (weightRule = WeightRulePart)?
;

[..]
```

Listing 4.1: Syntax der groben Struktur einer Geschäftsregel

Listing 4.1 zeigt zuerst die Grammatik für die grobe Struktur einer Geschäftsregel. Beginnend mit dem Präfix `RULE` kann eine neue Geschäftsregel angelegt werden, welche durch eine Identifikation (`id`) und eine Beschreibung (`title`) eingeführt wird. Diese beiden Terminale können Werte vom Typ `String` annehmen, wobei eine Identifikation verpflichtend ist und eine Beschreibung optional (definiert durch das Zeichen '?') über das Schlüsselwort `description` hinzugefügt werden kann. Des Weiteren führt die Grammatik die einzelnen Teile einer Geschäftsregel durch die Nicht-Terminale `GivenRulePart, WhenRulePart, ThenRulePart` und `WeightRulePart` ein, wobei die ersten drei obligatorisch sind und das letzte optional ist. Hierdurch steht es einem Experten frei, einen entsprechenden Gewichtungsfaktor für eine Geschäftsregel zu spezifizieren.

Die dargestellten Nicht-Terminale sind im Rahmen der Grammatik weiter definiert. Diese Detaillierung unserer DSL-Syntax wird am Beispiel des `GIVEN`-Teils näher erläutert. Die formale Syntax der weiteren Geschäftsregel-Teile kann dem Anhang entnommen werden (vgl. Abschnitt A.1).

In Listing 4.2 wird die Formalisierung des Nicht-Terminals `GivenRulePart` gezeigt. Dieses kann durch das Präfix `GIVEN` eingeführt werden und beinhaltet die beiden Nicht-Terminale `InformationElement` und `ApplicationCondition`. Ersteres beschreibt die Informationen beziehungsweise Daten, die für die Anwendung einer Regel verfügbar sein müssen (z.B. Variabilität, Kosten oder Funktionen einer Komponente)

```
[..]

GivenRulePart:
   'GIVEN' informationElement += InformationElement*
      'AND' applicationConditions += ApplicationCondition
      ('OR' applicationConditions += ApplicationCondition)*
;

[..]
```

Listing 4.2: Grammatikausschnitt für den `GivenRulePart`

und letzteres, auf welche Art von Kandidaten für unnötige Variabilität (vgl. Unterabschnitt 3.3.2) eine Regel angewendet wird. Diese beiden Nicht-Terminale sind durch einen `AND`-Operator miteinander zu verknüpfen. Weiterhin können beide beliebig oft (null bis unendlich, definiert durch das Zeichen '*') hintereinander verwendet werden. Hierfür werden die entsprechenden DSL-Features `informationElement` und `applicationConditions` mit Hilfe des Zuweisungsoperators `+=` als Liste definiert. Zudem können mehrere Nicht-Terminale `ApplicationCondition` definiert und durch Verknüpfung mit einem `OR`-Operator zusammen verwendet werden. So ist eine Geschäftsregel dann anzuwenden, wenn ein betrachteter Kandidat mindestens eine dieser Konditionen erfüllt. Beide genannten Nicht-Terminale werden nachfolgend detaillierter beschrieben.

Listing 4.3 zeigt die detaillierte Formalisierung für das Nicht-Terminal `InformationElement`. Es wird durch die beiden Schlüsselwörter `'information on'` eingeführt und enthält weitere Nicht-Terminale vom Typ `NeededInformationElement`. Diese können, durch ein Komma getrennt, aneinandergereiht werden, wobei mindestens eines beschrieben werden muss und

```
[..]

InformationElement:
   'information' 'on' neededInformation
      += NeededInformationElement
      (',' neededInformation += NeededInformationElement)*
;

NeededInformationElement:
   VariabilityInformationElement
      | UsageFrequencyInformationElement
      | FeatureInformationElement
      | DependencyInformationElement
      | StrategicFitInformationElement
      | TCOInformationElement
      | RestructuringCostInformationElement
;

VariabilityInformationElement:
   {VariabilityInformationElement} 'variability'
;

UsageFrequencyInformationElement:
   {UsageFrequencyInformationElement} 'frequencies'
;

[..]
```

Listing 4.3: Grammatikausschnitt für das `InformationElement`

weitere hinten angestellt werden können. `NeededInformationElement` beschreibt alle Informationen, die über einen Kandidaten zur Verfügung stehen müssen. Hierzu zählen beispielsweise seine Variabilität, Nutzungshäufigkeit, TCO oder Funktionen. Zu diesem Zweck bietet `NeededInformationElement` eine alternative Auswahl an weiteren Nicht-Terminalen an (definiert durch das Zeichen '`|`'), z.B. das `VariabilityInformationElement`, welches Informationen über die Variabilität eines betrachteten Kandidatens benötigt, oder das `UsageFrequencyInformationElement`, welches die Nutzungshäufigkeit eines Kandidatens erforderlich macht. Alle Nicht-Terminale, die alternativ zur Auswahl stehen, werden durch weitere Nicht-Terminale definiert. So sind beispielsweise `VariabilityInformationElement` und `UsageFrequencyInformationElement` durch die beiden Schlüsselwörter `'variability'` und `'frequencies'` definiert. Ihnen vorangestellt ist ein Attribut (z.B. `{VariabilityInformationElement}`), welches dafür sorgt, dass ein entsprechendes DSL-Feature in der abstrakten Syntax (dem Metamodell) der DSL für das jeweilige Schlüsselwort erzeugt wird.

In Listing 4.4 ist die weitere Formalisierung des Nicht-Terminals `ApplicationCondition` dargestellt. Dieses stellt die Auswahl der beiden Alternativen `SingleApplicationCondition` und `GroupApplicationCondition` zur Verfügung, welche ebenfalls Nicht-Terminale darstellen. Während `SingleApplicationCondition` durch das Schlüsselwort `'singleElement'` definiert ist, wird `GroupApplicationCondition` mit Hilfe des Schlüsselwortes `'variabilityGroup'` eingeführt und mit dem anschließenden Nicht-Terminal `GroupVariability` beschrieben. Dieses ist durch die beiden alternativen Nicht-Terminale `AlternativeGroupVariability` und `RelatedGroupVariability` definiert, welche durch die beiden Schlüsselwörter `'alternative'` und `'related'` spezifiziert sind.

Mit Hilfe der vorgeschlagenen DSL können Domänenexperten Regeln erstellen, um damit ihre Geschäftsanforderungen und -ziele zu beschreiben. Hierbei können sie verschiedene Kriterien, wie z.B. die Variabilität, die Nutzungshäufigkeit, die Kosten, die Funktionen oder den strategischen Fit eines Kandidaten für unnötige Variabilität im `WhenRulePart` der DSL berücksichtigen (vgl. Listing A.5). Zudem kann die DSL um eigene individuelle Kriterien erweitert werden. Hierzu bietet unsere konkrete Syntax das Nicht-Terminal `CustomCharacteristicElement` (vgl. Listing A.9), welches eine frei definierbare Zeichenkette in Form eines String-Terminals aufnehmen kann. Darüber hinaus ist die DSL darauf ausgelegt, dass solche Regeln für Kandidaten sowohl auf Ebene ihres logischen Elementes, als auch auf Ebene ihres physischen Elementes spezifiziert werden können. So ist eine Berücksichtigung von Konfigurationsoptionen sowie von einzelnen Versionen einer Komponente im `WhenRulePart` und `ThenRulePart` der DSL möglich. Hierzu enthält unsere konkrete Syntax das Nicht-Terminal `ModelSelectionElement`, welches die beiden alternativen Nicht-Terminale `logicalCandidate` und `physicalCandidate` definiert (vgl. Listing A.10).

```
[..]

ApplicationCondition:
   SingleApplicationCondition
      | GroupApplicationCondition
;

SingleApplicationCondition:
   {SingleApplicationCondition} 'singleElement'
;

GroupApplicationCondition:
   {GroupApplicationCondition} 'variabilityGroup'
      groupVariability = GroupVariability
;

GroupVariability:
   AlternativeGroupVariability
      | RelatedGroupVariability
;

AlternativeGroupVariability:
   {AlternativeGroupVariability} 'alternative'
;

RelatedGroupVariability:
   {RelatedGroupVariability} 'related'
;

[..]
```

Listing 4.4: Grammatikausschnitt für die `ApplicationCondition`

In dem Listing 4.5 ist die formalisierte Beschreibung der beispielhaften Geschäftsregel `GR1` dargestellt, welche mit Hilfe unserer entworfenen DSL spezifiziert wurde. Diese Regel hat die Identifikation `GR1` und ist mit einer natürlichsprachigen Beschreibung versehen. Im `GIVEN`-Teil ist die Voraussetzung beschrieben, dass Informationen über die Variabilität und die Nutzungshäufigkeit eines Kandidaten

```
RULE "GR1" description "Standardize candidates in Variability
   groups with usage frequency greater than 66%"
GIVEN information on variability, frequencies AND
   variabilityGroup related OR variabilityGroup alternative
WHEN logicalCandidate has modelFrequency > 66
THEN logicalCandidate is required
WEIGHT 3
```

Listing 4.5: Spezifizierung der beispielhaften Geschäftsregel GR1

```
RULE "GR2" description "Remove independent optional candidates
   with usage frequency less than 10%"
GIVEN information on variability, frequencies AND singleElement
WHEN logicalCandidate has modelFrequency < 10
THEN logicalCandidate is notRequired
```

Listing 4.6: Spezifizierung der beispielhaften Geschäftsregel GR2

vorliegen müssen, damit die Geschäftsregel angewendet werden kann. Außerdem wird die Regel auch nur auf solche Kandidaten angewendet, die sich in einer Alternativ- oder Relationsgruppe befinden. Der `WHEN`-Teil bestimmt die Bedingung, dass die `modelFrequency`[2] des betrachteten Kandidatens über 66% liegen muss. Die daraus folgende Konsequenz ist im `THEN`-Teil beschrieben und besagt, dass der Kandidat in solch einem Fall als *erforderlich (required)* betrachtet wird. Zudem fließt diese Regel mit einem Gewicht von 3 in die Bewertung ein, was in dem `WEIGHT`-Teil definiert ist.

Listing 4.6 zeigt die Spezifizierung der beispielhaften Geschäftsregel `GR2`, welche ebenfalls über eine entsprechende Identifikation und Beschreibung verfügt. Im `GIVEN`-Teil dieser Regel ist definiert, dass Informationen über die Variabilität und die Nutzungshäufigkeit vorliegen müssen. Zudem muss es sich bei einem Kandidaten um eine unabhängige Konfigurationsoption handeln, was durch das Schlüsselwort `singleElement` spezifiziert ist. Sollte diese Konfigurationsoption eine `modelFrequency` aufweisen, die kleiner als 10% ist (`WHEN`-Teil), dann wird der entsprechende Kandidat als *nicht erforderlich (not required)* betrachtet (`THEN`-Teil).

Abschließend ist im Listing 4.7 auch die formale Spezifikation der beispielhaften Geschäftsregel `GR3` mit ihrer entsprechenden Identifikation und Beschreibung dargestellt. Für diese werden Informationen über die Variabilität und die TCO eines Kandidaten für unnötige Variabilität benötigt, welcher einer Variabilitätsgruppe zugeordnet sein muss (`GIVEN`-Teil). Sollte der betrachtete Kandidat dabei den geringsten TCO-Wert in seiner Variabilitätsgruppe aufweisen (`WHEN`-Teil), dann wird dieser als *erforderlich (required)* betrachtet (`THEN`-Teil).

```
RULE "GR3" description "Standardize candidates in Variability
   groups with lowest TCO"
GIVEN information on variability, tco AND variabilityGroup
   related OR variabilityGroup alternative
WHEN logicalCandidate has tco = minmum
THEN logicalCandidate is required
```

Listing 4.7: Spezifizierung der beispielhaften Geschäftsregel GR3

[2] In unserer DSL wird die Häufigkeit H_{ko_a} (vgl. Gleichung 3.2) als *modelFrequency* und die Häufigkeit $H_{k_j(ko_a)}$ (vgl. Gleichung 3.3) als *individualFrequency* bezeichnet (vgl. Listing A.9)

4.1.2 Regelbasierte Analyse von Kandidaten für unnötige Variabilität

Basierend auf den beschriebenen Geschäftsregeln kann nun eine regelbasierte Analyse der Kandidaten für unnötige Variabilität durchgeführt werden, um diese aus einer businessorientierten Perspektive zu bewerten. Dabei ist das Ziel dieser Bewertung, zu analysieren, in welchem Umfang die einzelnen Kandidaten die beschriebenen Geschäftsanforderungen und -ziele unterstützen. Hierdurch lässt sich feststellen, ob bestimmte technologische Komponenten im Einsatz sind, die aus einer businessorientierten Sicht nicht erforderlich sind, da sie die gestellten Geschäftsanforderungen weniger erfüllen, als andere Kandidaten. Zu diesem Zweck werden die aus einem 150%-Modell abgeleiteten Kandidaten für unnötige Variabilität (vgl. Abschnitt 3.3) für die weitere Analyse herangezogen. Um unseren regelbasierten Ansatzes in einer geeigneten Art und Weise zu veranschaulichen, wird hierfür das fortlaufende Beispiel aus Kapitel 3 wieder aufgegriffen.

In einem ersten Schritt werden die zusammenhängenden Kandidaten aus dem Set aller Kandidaten für unnötige Variabilität (vgl. Unterabschnitt 3.3.2) extrahiert, um sie für die Analyse einzeln betrachten zu können. Dabei sind Kandidaten dann *zusammenhängend*, wenn sie derselben Variabilitätsgruppe zugeordnet sind und sich damit auch in derselben Intersection der analysierten TAPs befinden (vgl. Abbildung 3.9). Basierend auf dem Set aller Kandidaten aus unserem fortlaufenden Beispiel (vgl. Tabelle 3.14) extrahieren wir die Alternativgruppe `AG1((A1) Tomcat, (A2) WebSphere App. Server))` mit den beiden Kandidaten `(A1) Tomcat` und `(A2) WebSphere App. Server` für unsere regelbasierte Analyse (vgl. Tabelle 4.1).

Im nächsten Schritt werden alle verfügbaren Geschäftsregeln nacheinander auf die extrahierten Kandidaten angewendet. Hierfür wird als erstes überprüft, ob ein Kandidat die Voraussetzungen einer Geschäftsregel erfüllt, damit diese auf den Kandidaten angewendet werden kann. Diese Überprüfung erfolgt anhand des `GIVEN`-Teils einer Geschäftsregel. Hier kann beschrieben sein, dass spezifische Informationen über einen Kandidaten vorliegen müssen und der betrachtete Kandidat einem ausgewählten Typ entsprechen soll. Beispielhaft lässt sich diese Überprüfung anhand der exemplarischen Geschäftsregel `GR1` aus dem Listing 4.5 verdeutlichen. Diese Regel erwartet Informa-

Variabilität		Kandidaten	TAP
AG1	A1	Tomcat	1, 2, 4
	A2	WebSphere App. Server	3

AG = Alternativgruppe *A* = alternativ

Tabelle 4.1: Extrahierte Kandidaten für die regelbasierte Analyse des fortlaufenden Beispiels

tionen über die Variabilität und die Nutzungshäufigkeit des betrachteten Kandidaten sowie, dass dieser einer Alternativ- oder Relationsgruppe zugeordnet sein muss. Da diese Informationen für unsere betrachteten Beispielkandidaten `(A1) Tomcat` und `(A2) WebSphere App. Server` im 150%-Modell vorliegen und es sich dabei um eine Alternativgruppe (`AG1`) handelt, sind alle Voraussetzungen der Geschäftsregel erfüllt und sie kann auf die ausgewählten Kandidaten angewendet werden. Wären die entsprechenden Voraussetzungen nicht erfüllt, dann würde die Geschäftsregel nicht zur Anwendung kommen. Dabei wird eine Geschäftsregel immer auf alle zusammenhängenden Kandidaten einer Variabilitätsgruppe angewendet oder auf keinen, falls die Voraussetzungen nicht erfüllt sind.

Als nächstes wird die Bedingung einer Geschäftsregel, die im `WHEN`-Teil beschrieben ist, im Hinblick auf die betrachten Kandidaten untersucht. Dabei steht die Frage im Vordergrund, ob die jeweiligen Kandidaten die definierte Bedingung erfüllen oder nicht. In unserer exemplarischen Geschäftsregel `GR1` (vgl. Listing 4.5) ist die Bedingung beschrieben, dass die Nutzungshäufigkeit H_{ko_a} einer Konfigurationsoption mehr als 66% betragen soll. Für die beiden Kandidaten aus unserem fortlaufenden Beispiel ergeben sich hierfür die Werte H_{ko_a}=75% (`Tomcat`) und H_{ko_a}=25% (`WebSphere App. Server`). Daraus resultiert, dass der Kandidat `Tomcat` die Bedingung der Geschäftsregel erfüllt und der Kandidat `WebSphere App. Server` die definierte Bedingung nicht erfüllt.

Die Konsequenz, die sich daraus ergibt, ist im `THEN`-Teil einer Geschäftsregel beschrieben und führt dazu, dass ein Kandidat entweder als *erforderlich (required)* oder als *nicht erforderlich (not required)* klassifiziert wird. Im Fall einer Klassifizierung als *erforderlich*, wird dem betrachteten Kandidaten der *Wert 1* und im umgekehrten Fall der *Wert 0* zugeordnet. Für diese Werte kann zusätzlich ein Gewichtungsfaktor, der im `WEIGHT`-Teil einer Geschäftsregel definiert ist, berücksichtigt werden. In unserer beispielhaften Geschäftsregel `GR1` (vgl. Listing 4.5) wird ein Kandidat, der die genannte Bedingung erfüllt, als *erforderlich* betrachtet. Dies hat zur Folge, dass der Kandidat `Tomcat` den Wert 1 und der Kandidat `WebSphere App. Server` den Wert 0 zugeordnet bekommt. Aufgrund des Gewichtungsfaktors von 3, der in unserer Geschäftsregel beschrieben ist, werden die genannten Werte noch mit diesem Faktor multipliziert. So ergibt sich im Endeffekt der Wert 3 für den Kandidaten `Tomcat` und der Wert 0 für den Kandidaten `WebSphere App. Server`. Dies stellt das Bewertungsergebnis der Geschäftsregel `GR1` für die beiden zusammenhängenden Kandidaten `AG1((A1) Tomcat, (A2) WebSphere App. Server))` dar.

Im letzten Schritt wird eine *Bewertungsmatrix* für jede Gruppe zusammenhängender Kandidaten erstellt. Diese Matrix enthält alle Bewertungsergebnisse der einzeln angewendeten Geschäftsregeln für die entsprechenden Kandidaten. Durch Aufsummierung der jeweiligen Einzelergebnisse wird eine Gesamtbewertung für jeden Kandidaten errechnet. Diese zeigt, wie gut beziehungsweise in welchem Umfang ein Kandidat die gestellten Geschäftsanforderungen und -ziele erfüllt. In Tabelle 4.2 ist die Bewertungsmatrix für unsere ausgewählten Kandidaten beispielhaft dargestellt.

Variabilität		Kandidat	GR1	GR3	GR4	Gesamt
AG1	A1	Tomcat	3	1	0	4
	A2	WebSphere App. Server	0	0	1	1

AG = Alternativgruppe A = alternativ

Tabelle 4.2: Bewertungsmatrix für die Kandidaten des fortlaufenden Beispiels

Wie anhand der Matrix in Tabelle 4.2 zu erkennen ist, wurden die exemplarischen Geschäftsregeln `GR1, GR3` und `GR4` auf die ausgewählten Kandidaten `Tomcat` und `WebSphere App. Server` angewendet. Während `GR1` und `GR3` bereits als beispielhafte Regeln beschrieben worden sind (vgl. Unterabschnitt 4.1.1), ist die spezifizierte Geschäftsregel `GR4` im folgenden Listing 4.8 dargestellt.

```
RULE "GR4" description "Standardize candidates in Variability
   groups with highest number of Features"
GIVEN information on variability, features AND variabilityGroup
    related OR variabilityGroup alternative
WHEN logicalCandidate has feature = maximum
THEN logicalCandidate is required
```

Listing 4.8: Spezifizierung der beispielhaften Geschäftsregel GR4

Die Geschäftsregel `GR4` hat den Zweck, solche Kandidaten zu standardisieren, die den größten Funktionsumfang innerhalb ihrer Variabilitätsgruppe aufweisen. Um dies zu spezifizieren, wird im `WHEN`-Teil der Regel der Ausdruck `feature = maximum` verwendet. In dem hier gezeigten Beispiel trifft dies auf den Kandidaten `WebSphere App. Server` zu, sodass dieser mit dem Wert 1 und der Kandidat `Tomcat` im Umkehrschluss mit dem Wert 0 versehen wird.

Die Bewertungsmatrix in Tabelle 4.2 enthält alle Einzelergebnissen, die sich aus der Anwendung der jeweiligen Geschäftsregeln ergeben. Zudem wird hierfür auch die resultieren Gesamtbewertung bestimmt, nach welcher der Kandidat `Tomcat` den Gesamtwert 4 und der Kandidat `WebSphere App. Server` den Gesamtwert 1 erhält. Demnach erfüllt der Kandidat `Tomcat` die gestellten Geschäftsanforderungen und -ziele besser als der Kandidat `WebSphere App. Server`.

4.1.3 Ableitung von Restrukturierungspotentialen

Zur Bestimmung von Elementen, die tatsächlich zu unnötiger Variabilität führen, müssen zuerst die Ergebnisse aus der vorherigen businessorientierten Analyse interpretiert werden. Das Ziel dabei ist, solche Kandidaten zu identifizieren, die aus einer businessorientierten Sicht für unnötige Variabilität sorgen, weil sie die Geschäftsanforderungen und -ziele weniger gut unterstützen als andere Kandidaten.

Zu diesem Zweck werden *Restrukturierungspotentiale* für die bewerteten Kandidaten auf Basis ihrer entsprechenden Bewertungsmatrix identifiziert.

Definition 4.3: Restrukturierungspotential

Unter einem *Restrukturierungspotential (RP)* verstehen wir eine mögliche Umstrukturierung von TAPs mit dem Ziel, einen konkreten Kandidaten, der im Rahmen einer businessorientierten Analyse als nicht erforderlich bewertet wurde, zu eliminieren. Da eine Analyse aus technischer Sicht hierbei noch keine Berücksichtigung findet, stellen diese möglichen Modifikationen erst einmal nur Potentiale dar.

Solche Restrukturierungspotentiale können Maßnahmen zur *Ersetzung* oder *Entfernung* eines bestimmten Elementes in den betroffenen TAPs darstellen. Dabei meint ein *Ersetzungspotential* `RPi(A ERSETZT B)`, dass ein *Kandidat B* aus den betroffenen TAPs entfernt und durch einen anderen *Kandidaten A* aus derselben Variabilitätsgruppe ersetzt wird. Da hierfür mindestens zwei verschiedene, zusammenhängende Kandidaten vorhanden sein müssen, können solche Restrukturierungspotentiale nur für Variabilitätsgruppen abgeleitet werden. Dagegen stellt ein *Entfernungspotential* `RPi(ENTFERNUNG C)` eine Maßnahme dar, welche einen *Kandidaten C* aus betroffenen TAPs entfernt, ohne diesen dabei durch ein anderes Element zu ersetzen. Solch ein Restrukturierungspotential wird daher nur für einzelne Konfigurationsoptionen abgeleitet, wenn diese eine Komponente beinhalten, der nur in geringem Maße verwendet wird und somit eine Einzellösung darstellt. Diese gilt es aus businessorientierter Sicht zu minimieren, da sie oft hohe Kosten für Wartung und Weiterentwicklung verursachen. Darüber hinaus sind keine weiteren RP-Operatoren (z.B. *Hinzufügen*) notwendig, um einen Kandidaten, der als nicht erforderlich bewertet wurde, aus den betroffenen TAPs zu eliminieren. Dies ist nur durch Ersetzung oder Entfernung möglich.

Zur Ableitung von Restrukturierungspotentialen werden die Ergebnisse einer erstellten Bewertungsmatrix analysiert. Beinhaltet solch eine Matrix eine Variabilitätsgruppe, werden die Gesamtergebnisse der einzelnen Kandidaten miteinander verglichen, um mögliche Ersetzungen zu identifizieren. Für jeden Kandidaten, der eine geringere Gesamtbewertung als ein anderer Kandidat innerhalb solch einer Gruppe hat, wird ein Ersetzungspotential durch den höherwertigen Kandidaten abgeleitet. So kann beispielsweise für unsere exemplarische Bewertungsmatrix in Tabelle 4.2 das Restrukturierungspotential `RP1(Tomcat ERSETZT WebSphere App. Server)` ableiten werden. Diese mögliche Ersetzung ergibt sich daraus, dass für den Kandidaten `Tomcat` der Gesamtwert 5 und für den Kandidaten `WebSphere App. Server` der Gesamtwert 1 ermittelt wurde.

Beinhaltet eine generierte Bewertungsmatrix dagegen eine einzelne Konfigurationsoption, so wird untersucht, ob hierfür eine mögliche Entfernung abgeleitet werden kann. Auch dies wird anhand des Gesamtergebnisses des betrachteten Kandidaten

festgestellt. Weist dieser einen Gesamtwert von 0 auf, so wird hierfür ein Entfernungspotential generiert. Dies stellt den kleinstmöglichen Wert für einen Kandidaten dar und zeigt an, dass dieser durch keine einzige der angewendeten Regeln als erforderlich bewertet worden ist. Da dies auf eine Einzellösung hindeuten kann, sollte diese durch einen Domänenexperten näher analysiert werden. Eine bloße Entfernung solch einer Konfigurationsoption ist dabei nicht ohne weiteres möglich. Daher sollte zuerst überprüft werden, ob es eine andere Komponente gibt, welche die entsprechende Funktion übernehmen kann. Alternativ kann auch außerhalb der betrachteten TAPs nach einer weiteren Komponente gesucht werden, die eine ähnliche Funktionalität bietet, sodass eine entsprechende Ersetzung für den betroffenen Kandidaten realisiert werden kann.

Demnach muss festgehalten werden, dass mit der Generierung von solchen Entfernungspotentialen das Ziel verfolgt wird, Domänenexperten auf mögliche Einzellösungen hinzuweisen und die Experten im Rahmen der späteren Bewertung (vgl. Kapitel 5) bei der Identifizierung möglicher Ersetzungen zu unterstützen. Voraussetzung für die Generierung von Entfernungspotentialen ist allerdings, dass ein Domänenexperte mindestens eine Geschäftsregel spezifiziert hat, mit der ein Schwellwert S_E für die Nutzungshäufigkeit von solchen Einzellösungen definiert worden ist (z.B. 10%). Erst wenn diese Regel auf den betrachteten Kandidaten angewendet worden ist und sein Gesamtwert danach bei 0 liegt, wird er als nicht erforderlich betrachtet, sodass ein entsprechendes Entfernungspotential für diesen Kandidaten abgeleitet werden kann. Beispielhaft dafür kann das Restrukturierungspotential `RP2(ENTFERNUNG phpMyAdmin)` genannt werden, welches aus den Kandidaten für unnötige Variabilität aus unserem fortlaufenden Beispiel (vgl. Tabelle 3.14) abgeleitet werden würde, wenn der Schwellwert S_E für unser Beispiel bei 50% liegen würde.

Design-Entscheidung 4.1: Schwellwert S_E für Entfernungspotentiale

Der Schwellwert S_E für die Ableitung von Entfernungspotentialen für Kandidaten, die eine unabhängige Konfigurationsoption darstellen, ist über eine entsprechende Geschäftsregel frei konfigurierbar. In Gesprächen mit Experten und der Arbeit mit den Daten unseres Industriepartners hat sich gezeigt, dass hier ein Wert von 10% ($S_E = 0,1$) angemessen ist.

Da höher bewertete Kandidaten die gestellten Geschäftsanforderungen und -ziele besser erfüllen als niedriger bewertete, werden solche Kandidaten aus businessorientierter Sicht als Möglichkeit für eine Standardisierung in Betracht gezogen. Es kann für Domänenexperten allerdings auch von Interesse sein, gleichwertige Kandidaten gegeneinander abzuwägen, um auch deren Variabilität zu reduzieren und lediglich eines oder wenige dieser Element als zukünftigen Standard zu nutzen. Aus diesem Grund können mit Hilfe unseres Ansatzes auch Ersetzungspotentiale für Kandidaten abgeleitet werden, die den gleichen Gesamtwert in der Bewertungsmatrix aufweisen. Da diese allerdings weniger bedeutsam

sind als Ersetzungspotentiale für verschieden bewertete Kandidaten, nutzen wir einen *Confidence-Wert* co_{RP}, der die Bedeutsamkeit unterschiedlicher Restrukturierungspotentiale für Domänenexperten sichtbar macht. So wird der Confidence-Wert $co_{RP} = stark$ jedem Ersetzungspotential zugewiesen, welches den Austausch eines Kandidaten durch einen höherwertigen Kandidaten beabsichtigt. Im Gegensatz dazu erhalten Ersetzungspotentiale für gleichwertige Kandidaten den Confidence-Wert $co_{RP} = schwach$ zugewiesen. Demnach ergibt sich für unser exemplarisches Ersetzungspotential `RP1(Tomcat ERSETZT WebSphere App. Server)` der Confidence-Wert $co_{RP} = stark$ (vgl. Tabelle 4.2).

4.2 Analyse von Restrukturierungspotentialen aus technischer Sicht

Die in dem vorherigen Abschnitt vorgestellte regelbasierte Businessanalyse unterstützt Experten dabei, Kandidaten für unnötige Variabilität aus einer businessorientierten Perspektive zu bewerten und daraus Restrukturierungspotentiale abzuleiten. Diese sind allerdings bisher nicht aus Sicht einer technologieorientierte Analyse bewertet worden, sodass es vorgeschlagene Restrukturierungspotentiale geben kann, welche aufgrund von technischen Limitierungen schwierig oder gar nicht umsetzbar sind. Hierzu kann es beispielsweise kommen, wenn ein Ersetzungspotential den Austausch einer Komponente durch eine andere Komponente vorsieht und dafür zu viele beteiligte Komponenten angepasst werden müssten. Daher präsentieren wir in diesem Abschnitt die *Graphbasierte Technologieanalyse* (vgl. Abbildung 4.3), welche die Bewertung der technischen Machbarkeit von Restrukturierungspotentialen erlaubt.

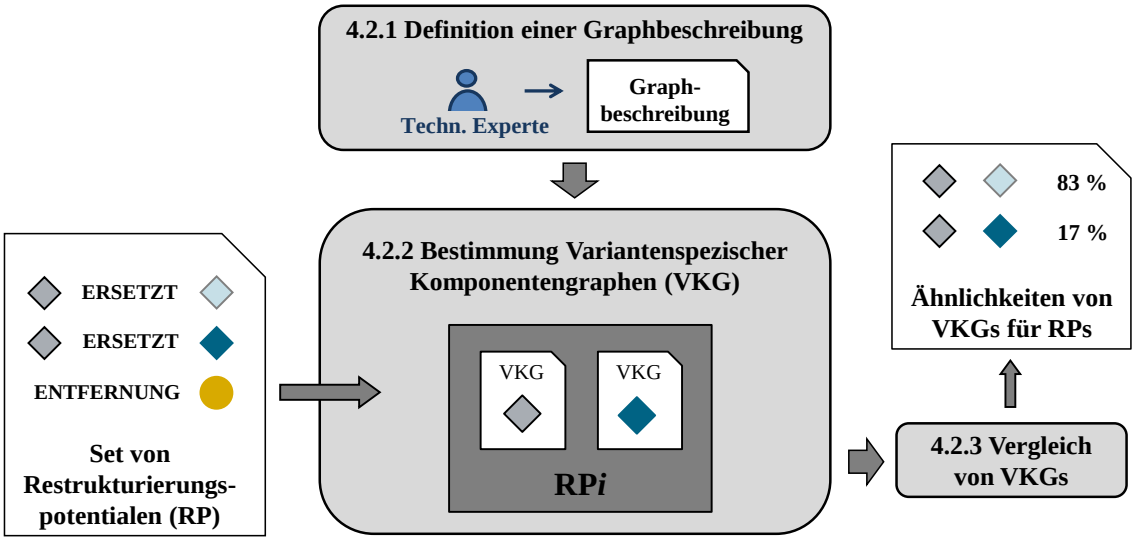


Abbildung 4.3: Workflow für die graphbasierte Technologieanalyse

Wie in Abbildung 4.3 dargestellt ist, wird im Rahmen unseres Ansatzes zuerst eine Graphbeschreibung durch einen Technologieexperten definiert (vgl. Unterabschnitt 4.2.1). Hierdurch können technische Anforderungen spezifiziert und so für die Analyse berücksichtigt werden. Basierend darauf können dann die variantenspezifischen Komponentengraphen (VKGs) der jeweiligen Kandidaten eines Restrukturierungspotentials bestimmt werden (vgl. Unterabschnitt 4.2.2). Solche VKGs beinhalten alle Komponenten, die mit dem betrachteten Kandidaten in Beziehung stehen, da sie mit diesem gemeinsam in einem oder mehreren TAPs verbaut sind. Anschließend können die einzelnen VKGs miteinander verglichen werden, um ihre Ähnlichkeit zu bestimmen (vgl. Unterabschnitt 4.2.3). Hierdurch lässt sich feststellen, wie viele der beteiligten Komponenten bereits gemeinsam mit allen betrachteten Kandidaten eines Restrukturierungspotentials genutzt werden, sodass es möglich ist, die technische Machbarkeit des analysierten Restrukturierungspotentials einzuschätzen.

4.2.1 Graphbeschreibung zur Abbildung technischer Anforderungen

Um die technische Machbarkeit von Restrukturierungspotentialen bewerten zu können, wird eine graphbasierte Vergleichsanalyse der betroffenen Kandidaten durchgeführt. Zu diesem Zweck wird jeweils ein *variantenspezifischer Komponentengraph* für jeden einzelnen Kandidaten erstellt.

Definition 4.4: Variantenspezifischer Komponentengraph

Ein *Variantenspezifischer Komponentengraph (VKG)* beschreibt einen sternförmigen Graphen, welcher in der Mitte einen zentralen Knoten aufweist, der einen Kandidaten aus einem betrachteten Restrukturierungspotential darstellt. Alle mit diesem Kandidaten verbauten Komponenten (über alle TAPs) werden als weitere Knoten modelliert und durch gewichtete Kanten mit dem zentralen Knoten verbunden. Dabei drückt das Gewicht aus, in wie vielen TAPs eine Komponente zusammen mit dem betrachteten Kandidaten verbaut ist.

Die folgende Abbildung 4.4 stellt den VKG für einen *Kandidaten A* schematisch dar, der in zwei TAPs (signalisiert durch die Zahl in Klammern) eingesetzt wird. In diesem Beispiel ist der Kandidat A in beiden TAPs gemeinsam mit der `Komponente 2` und der `Komponente 3` verbaut. Die `Komponente 1` ist dagegen nur in einem TAP implementiert, welcher auch den betrachteten Kandidaten A beinhaltet.

Für die spätere Vergleichsanalyse solcher VKGs (vgl. Unterabschnitt 4.2.3) ist es sinnvoll, die technischen Anforderungen von Experten für diese Analyse mit einfließen zu lassen. So können sie beispielsweise definieren, ab welcher Nutzungshäufigkeit die technologischen Komponenten für einen Vergleich in Betracht gezogen werden, damit Einzellösungen die Bewertung der technischen Machbarkeit von Restrukturierungspotentialen nicht negativ beeinflussen. Somit haben technische Anforderungen

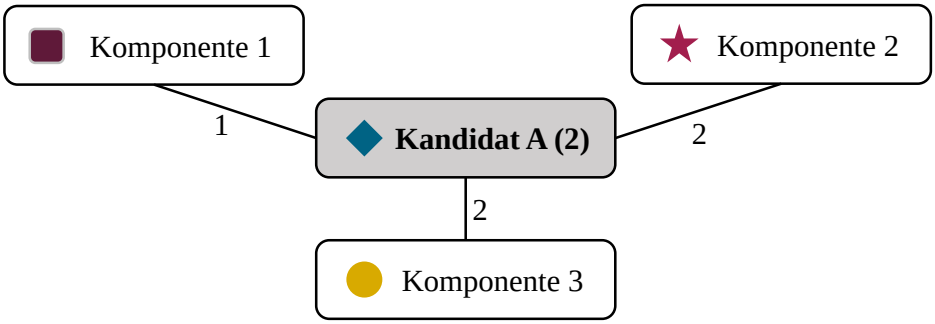


Abbildung 4.4: Schematische Darstellung eines variantenspezifischen Komponentengraphs für den Kandidaten A

direkten Einfluss auf die Analyse von VKGs und sollten bei deren Erstellung (vgl. Unterabschnitt 4.2.2) bereits berücksichtigt werden.

Damit die technischen Anforderungen von Experten berücksichtigt und automatisiert verarbeitet werden können, müssen sie formal beschrieben werden. Hierzu ist es erforderlich, einen benutzerfreundlichen Ansatz zu wählen, der die formale Beschreibung von Anforderungen mit wenig manuellem Aufwand erlaubt und dabei die Terminologie von Experten berücksichtigt. Aus diesem Grund haben wir eine weitere DSL entwickelt, welche mehr auf die Domäne von technischen Experten fokussiert ist und diese bei der Definition und formalen Beschreibung ihrer Anforderungen für die technische Analyse von VKGs unterstützt. Zu diesem Zweck können Experten mit Hilfe unserer DSL eine Graphbeschreibung erstellen, welche dann zur Laufzeit unseres Ansatzes dafür werdet wird, die identifizierten VKGs nach ihren Anforderungen zu erstellen. Da die vorgeschlagene DSL ein ähnliches Design wie unsere DSL für Geschäftsregeln aufweist (vgl. Unterabschnitt 4.1.1), kann ein Experte auch beide Rollen (business- und technologieorientiert) einnehmen und entsprechende Anforderungen mit Hilfe der beiden DSLs beschreiben.

Domain Specific Language zur Formalisierung von Graphbeschreibungen

Zur Formalisierung von technischen Anforderungen in Form einer Graphbeschreibung haben wir gemeinsam mit Experten unseres Industriepartners eine konkrete Syntax für unsere DSL entworfen. Die Struktur dieser Syntax besteht dabei aus vier Teilen, die jeweils durch einen spezifischen Präfix eingeführt werden:

1. *`GRAPH DESCRIPTION`*: [Titel der Graphbeschreibung]
2. *`ANALYZE`*: [Definition der zu analysierenden Layer und Tiers]
3. *`BUILD`*: [Instruktion für die Bildung von Graphen]
4. *`CORE`*: [Definition der Kerntechnologie-Elemente]

Mit dem Präfix `GRAPH DESCRIPTION` wird ein beschreibender Titel für eine Graphbeschreibung definiert. Zwar sieht unsere Vergleichsanalyse nur eine Graphbeschreibung zur Laufzeit vor, jedoch können so unterschiedliche Beschreibung angelegt und für die graphbasierte Analyse ausgewählt werden. Durch den Präfix `ANALYZE` kann definiert werden, welche Layer und Tiers der betrachteten TAPs für die Erstellung von Graphen berücksichtigt werden sollen. Hierdurch ist es möglich, den Fokus einer Analyse auf bestimmte Intersections zu legen. Mit Hilfe des Präfixes `BUILD` können Instruktionen zur Erstellung von Graphen beschrieben werden. Dabei kann vor allem bestimmt werden, unter welchen Voraussetzungen eine Komponente dem Graphen hinzugefügt wird (z.B. bei Überschreitung eines Schwellwertes für die Nutzungshäufigkeit). So können beispielsweise Einzellösungen bei Bedarf unberücksichtigt bleiben. Mit dem letzten Präfix `CORE` kann zudem bestimmt werden, welche der Komponenten eines erstellten Graphs als Kerntechnologie betrachtet werden. Dies erfolgt anhand der Kategorie von Komponenten (z.B. *Applikationsserver* oder *Relationales Datenbanksystem*). Dadurch lassen sich aus Expertensicht wichtige technologische Komponenten eines Graphens von weniger wichtigen unterscheiden, da nicht jede Komponente die gleiche Auswirkung auf die technische Machbarkeit von Restrukturierungen hat.

Die nachfolgenden Listings 4.9 bis 4.12 zeigen die Grammatik, also die konkrete Syntax, für die entwickelte DSL anhand von Auszügen. Auch in diesem Fall wurde die vorgeschlagene Syntax mit der Notation von Xtext umgesetzt (vgl. Abschnitt 2.3).

In Listing 4.9 ist die Syntax der 4-teiligen Struktur unserer DSL dargestellt. Wie hier zu erkennen ist, wird eine neue Graphbeschreibung mit dem Präfix `GRAPH DESCRIPTION` eingeführt, gefolgt von einer Identifikation (`id`) vom Typ String. Anschließend werden die weiteren drei Teile mit ihren entsprechenden Nicht-Terminalen `AnalyzePart`, `BuildPart` und `CorePart` beschrieben. Während der `AnalyzePart` ein verpflichtender Bestandteil einer Graphbeschreibung ist, können die anderen beiden Teile optional spezifiziert werden (durch das Zeichen '`?`' definiert).

Im nächsten Listing 4.10 wird der Grammatikausschnitt für das Nicht-Terminal `AnalyzePart` unserer DSL für die Graphbeschreibung veranschaulicht. Dieser Teil wird durch den Präfix `ANALYZE` eingeführt und beinhaltet die beiden Nicht-Terminale

```
Graph:
   'GRAPH' 'DESCRIPTION' id = STRING
      analyze = AnalyzePart
      (build = BuildPart)?
      (core = CorePart)?
;

[..]
```

Listing 4.9: Syntax der groben Struktur einer Graphbeschreibung

```
[..]

AnalyzePart:
   'ANALYZE' layerElements = LayerElement
      'AND' tierElements = TierElement
;

LayerElement:
   'layer' layerInformation += LayerType
      (',' layerInformation += LayerType)*
;

enum LayerType:
   ALL = 'all' | Layer_HARDWARE = 'hardware'
      | Layer_OPERATING_SYSTEM = "operatingSystem"
      | Layer_DATA_STORAGE = 'data'
      | Layer_APPLICATION_SERVER = 'application'
      | Layer_PRESENTATION_COMPONENTS = 'presentation'
;

TierElement:
   'tier' tierInformation += TierType
      (',' tierInformation += TierType)*
;

enum TierType:
   ALL = 'all' | Tier_CLIENT_TIER = 'client'
      | Tier_SERVER_TIER = 'server'
;

[..]
```

Listing 4.10: Grammatikausschnitt für den `AnalyzePart`

`LayerElement` und `TierElement`, welche mit einem logischen `AND`-Operator miteinander verbunden werden. Das erste Element kann eine Liste (mit Hilfe des Zeichens '`+=`' definiert) beliebig vieler (durch das Zeichen '`*`' definiert) `LayerTypes` aufnehmen und das zweite eine Liste beliebig vieler `TierTypes`. Sowohl `LayerType` als auch `TierTyp` sind beides *Aufzählungstypen* (`enum`), welche als Nicht-Terminale eine Auswahl von konkreten Layern und Tiers zur Verfügung stellen (z.B. `hardware` und `client`). Darüber hinaus bieten sie jeweils auch das Schlüsselwort '`all`' an, mit dem alle Layer beziehungsweise alle Tiers ausgewählt werden können. Das Zeichen '`|`' gibt dabei an, dass es sich bei den Auswahlmöglichkeiten für den `LayerType` und den `TierType` um Alternativen handelt.

Die hier aufgeführten Typen stellen dabei eine Basis für die Spezifikation von Graphbeschreibungen dar. Es ist jedoch möglich, diese Typen auch für einen unternehmensindividuellen Kontext anzupassen oder zu erweitern, indem eine eindeutige Bezeichnung eines Layers oder Tiers geändert oder ein neuer Typ mit entsprechender Bezeichnung hinzugefügt wird (z.B. `Tier_DB_TIER = 'database'`).

```
[..]

BuildPart:
   {BuildPart} 'BUILD' 'graph' 'including'
      (allInformation = 'all')?
      (numberInformation = NumberElement)? 'components'
      ('with' 'modelFrequency' '>' frequency = Frequency)?
;

NumberElement:
   value = INT
;

Frequency:
   value = INT
;

[..]
```

Listing 4.11: Grammatikausschnitt für den `BuildPart`

Listing 4.11 zeigt den Ausschnitt aus der Grammatik unserer DSL für den `BuildPart` einer Graphbeschreibung, welcher durch den Präfix `BUILD` eingeführt wird, gefolgt von dem sprechenden Schlüsselwort `'graph including'`. Solche zusätzlichen Schlüsselwörter erleichtern den Experten die Erstellung und die Lesbarkeit von Graphbeschreibungen. Danach kann durch Verwendung des Schlüsselwortes '`all`' oder des Nicht-Terminals `NumberElement` die Anzahl an Komponenten ('`components`') spezifiziert werden, die für den Graphen berücksichtigt werden sollen. Wird mit `NumberElement` eine spezifische Anzahl (durch das Terminal `INT`) von Komponenten definiert, so werden hierfür die Elemente mit der höchsten Nutzungshäufigkeit ausgewählt. Sind also beispielsweise sechs Komponenten gemeinsam mit einem Kandidaten in den betrachteten TAPs verbaut und wird `NumberElement=5` gesetzt, so bleibt die Komponente mit der geringsten Nutzungshäufigkeit für den VKG unberücksichtigt. Die Definition solch einer Anzahl ist allerdings optional, sodass auch alle Komponenten betrachtet werden können, die eine Nutzungshäufigkeit (`modelFrequency`) über einem bestimmten Schwellwert haben. Dies kann mit einem konkreten Wert für das Terminal `INT` über das Nicht-Terminal `Frequency` definiert werden. Ist also zum Beispiel '`all components with modelFrequency > 20`' spezifiziert worden, werden alle Komponenten mit einer Nutzungshäufigkeit von über 20% für den VKG des jeweiligen Kandidaten berücksichtigt. Dies verhindert beispielsweise, dass die Bestimmung der Ähnlichkeit von zwei VKGs durch Einzellösungen zu stark beeinträchtigt wird.

Der letzte Teil `CorePart` unserer DSL für Graphbeschreibungen ist in Listing 4.12 dargestellt. Mit Hilfe des Präfixes `CORE` wird dieser Teil eingeführt, gefolgt von dem sprechenden Schlüsselwort '`includes categories`'. Danach wird eine beliebig lange

```
[..]

CorePart:
   {CorePart} 'CORE' 'includes' 'categories'
      categoryInformation += CATEGORY_ID
      (',' categoryInformation += CATEGORY_ID)*
;

terminal CATEGORY_ID returns ecore::EString:
    ('CAT-') ('0'..'9')+
;
```

Listing 4.12: Grammatikausschnitt für den `CorePart`

Liste von Kategorien mit dem Terminal `CATEGORY_ID` aufgeführt, welches durch einen regulären Ausdruck definiert ist. Dieser beginnt mit der Zeichenkette `'CAT-'`, gefolgt von einer beliebig langen Zahl mit Ziffern zwischen 0 und 9, wobei mindestens eine Ziffer anzugeben ist (durch das Zeichen '+' definiert). Mit Hilfe solch einer `CATEGORY_ID` kann so eine spezifische Komponentenkategorie (z.B. *Java Applikationsserver* oder *Hierarchisches Datenbanksystem*) als Kerntechnologie definiert werden. Dies erlaubt eine noch präzisere Bestimmung der Ähnlichkeit von zwei VKGs (vgl. Unterabschnitt 4.2.3).

Mit Hilfe der vorgeschlagenen DSL ist es Experten nun möglich, technische Anforderungen für die Vergleichsanalyse von VKGs zu formulieren und mit Hilfe einer Graphbeschreibung zu spezifizieren. Solch eine beispielhafte Graphbeschreibung ist in Listing 4.13 dargestellt. In dieser ist spezifiziert, dass die Layer *Hardware*, *Betriebssystem*, *Daten* und *Präsentation* sowie alle verwendeten Tiers analysiert werden. Basierend darauf sollen solche Graphen gebildet werden, welche alle Komponenten berücksichtigen, die eine *Nutzungshäufigkeit* von *über 10%* aufweisen. Zusätzlich dazu ist beschrieben, dass alle Komponenten, die den Kategorien *x86-Client-Hardware* (`CAT-43`) und *Windows-Client-Betriebssystem* (`CAT-45`) sowie den Kategorien *x86-Server-Hardware* (`CAT-51`) und *UNIX-Server-Betriebssystem* (`CAT-53`) entsprechen, als Kerntechnologie betrachtet werden.

Im nächsten Unterabschnitt 4.2.2 wird gezeigt, wie mit Hilfe dieser exemplarischen Graphbeschreibung ein konkreter VKG für unser fortlaufendes Beispiel generiert werden kann.

```
GRAPH DESCRIPTION "Exemplary Graph Description"
ANALYZE layer hardware, operatingSystem, data, presentation AND
    tier all
BUILD graph including components with modelFrequency > 10
CORE includes categories CAT-43, CAT-45, CAT-51, CAT-53
```

Listing 4.13: Beispielhafte Graphbeschreibung

4.2.2 Bestimmung von variantenspezifischen Komponentengraphen

Zur Bestimmung von spezifischen VKGs für jeden einzelnen Kandidaten aus einem betrachteten Restrukturierungspotential, stellen wir den folgenden, zweistufigen Ansatz vor. Dabei wird eine durch Technologieexperten erstellte Graphbeschreibung im zweiten Schritt verwendet, um konkrete VKGs zu erstellen.

In einem ersten Schritt wird ein einzelner *Gesamtgraph* $G = (V, E)$ für alle analysierten TAPs erstellt. Dieser beinhaltet Knoten $v_i \in V$, die jeweils eine bestimmte Komponente auf einer spezifischen Layer-Tier-Intersection repräsentieren sowie Kanten $e_i \in E$, welche die Beziehungen zwischen den Komponenten darstellen. Solche Kanten zwischen den einzelnen Knoten beschreiben die gemeinsame Verwendung von Komponenten in den betrachteten TAPs. Dabei repräsentiert eine gewichtete Kante $e_i = (v_1, v_2, f \in \mathbb{N})$ die gemeinsame Verwendung der Knoten v_1 und v_2 in f TAPs.

Hierbei werden die einzelnen Komponenten von ihren jeweiligen physischen Elementen, also eingesetzte Versionen, abstrahiert und auf Ebene des logischen Elementes als Knoten modelliert. So wird beispielsweise für die beiden Komponenten `phpMyAdmin 2.5` und `phpMyAdmin 4.6` auf der `Präsentation-Server-Intersection` aus unserem fortlaufenden Beispiel (vgl. Tabelle 3.1) nur ein Knoten v_1 = `phpMyAdmin` erstellt. Dies erlaubt einen versionsunabhängigen Vergleich der eingesetzten Komponenten zwischen verschiedenen VKGs. Hierdurch werden die beiden Komponenten `phpMyAdmin 2.5` und `phpMyAdmin 4.6` als gleich betrachtet, was zu einer höheren Ähnlichkeit der entsprechenden VKGs führt. Zudem wird so die Komplexität solch eines Graphens, welcher die Technologiearchitektur eines Unternehmens repräsentiert, deutlich reduziert [BLNS12]. Experten ist es dennoch möglich, ihr Domänenwissen über eventuell vorhandene Limitierungen aufgrund von Versionsunterschieden bei der späteren Verifizierung von identifizierten Abhängigkeiten mit einfließen zu lassen (vgl. Abschnitt 5.3).

In Abbildung 4.5 ist der Gesamtgraph für unser fortlaufendes Beispiel auszugsweise dargestellt. Dieser beinhaltet nicht alle Knoten und Kanten, um die Lesbarkeit zu gewährleisten. Wie zu sehen ist, sind in diesem Graphen die Komponenten aus den vier betrachteten TAPs sowie deren Beziehungen abgebildet worden. Daran lässt sich beispielsweise erkennen, dass die logische Komponente `Redhat` insgesamt drei mal zusammen mit der logischen Komponente `x86` verbaut ist.

Im zweiten Schritt können nun die VKGs für Kandidaten eines beliebigen Restrukturierungspotentials aus dem erstellten Gesamtgraphen abgeleitet werden. Zur Erstellung eines spezifischen $VKG_{v_i} = (V, E)$, als Teilgraph von G, wird zuerst für einen selektierten Kandidaten der entsprechende Knoten $v_i \in V$ im Gesamtgraph G identifiziert. In unserem exemplarischen Restrukturierungspotential `RP1(Tomcat ERSETZT WebSphere App. Server)` gibt es die beiden Kandidaten `Tomcat` und `WebSphere App. Server`, welche durch die beiden Knoten v_1 = `Tomcat` und v_2 = `WebSphere App. Server` in G repräsentiert werden. Wird also beispiels-

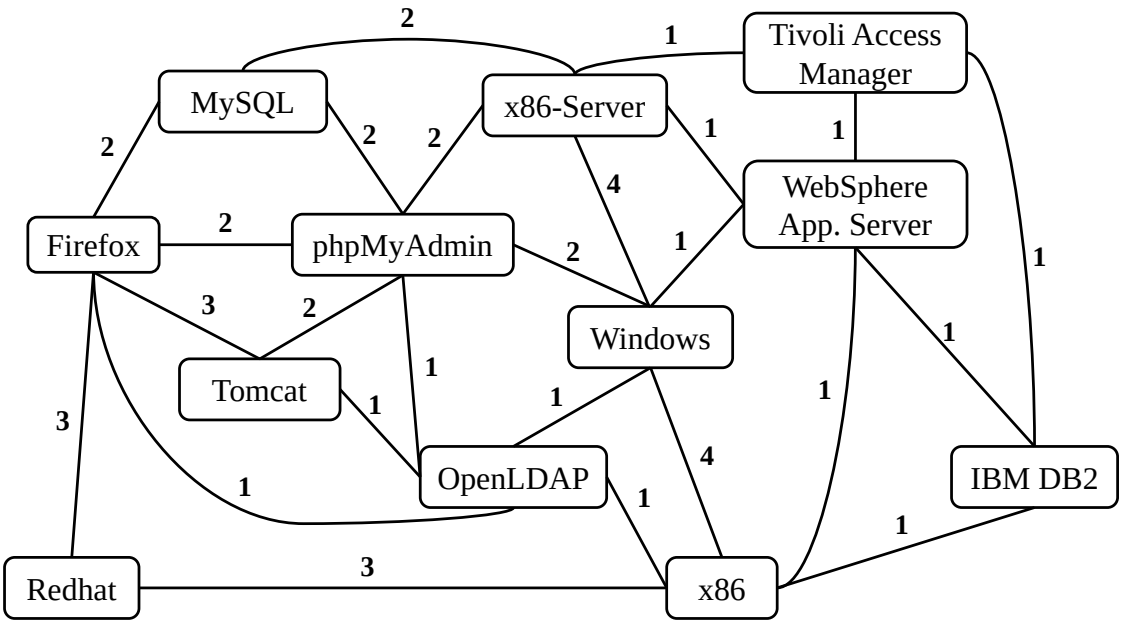


Abbildung 4.5: Gesamtgraph für das fortlaufende Beispiel (Ausschnitt)

weise der Kandidat `Tomcat` selektiert, so kann für diesen der VKG_{Tomcat} auf Basis des Gesamtgraphs G mit Hilfe der definierten Graphbeschreibung erstellt werden. Hierfür wird zuerst der Gesamtgraph G so beschnitten, dass alle Knoten, die keine direkte Beziehung zum betrachteten Knoten v_i haben, aus dem Graphen eliminiert werden. In unserem Beispiel würde dies unter anderem die Knoten v_3 = `IBM DB2` und v_4 = `Tivoli Access Manager` betreffen. Hierdurch entsteht bereits der zu berücksichtigende Teilgraph VKG_{v_i}, welcher allerdings noch auf Basis einer vorliegenden Graphbeschreibung an die technischen Anforderungen von Experten angepasst werden muss. Solch ein VKG beinhaltet damit alle Konfigurationsoptionen, die zusammen mit dem betrachteten Kandidaten in mindestens einer der gültigen Konfigurationen (den entsprechenden TAPs) verbaut sind. Da für die Bestimmung der technischen Machbarkeit eines Restrukturierungspotentials nur solche Konfigurationsoptionen eine Rolle spielen, die gemeinsam mit den betrachteten Kandidaten in einer technischen Systemarchitektur verwendet werden, sind transitive Beziehungen innerhalb des Gesamtgraphs G nicht relevant.

Um solch einen generierten VKG anschließend im Hinblick auf die individuellen technischen Anforderungen anzupassen, wird eine von Experten spezifizierte Graphbeschreibung benötigt. Für unser fortlaufendes Beispiel ziehen wir dazu die exemplarische Graphbeschreibung aus dem Listing 4.13 heran.

Der `ANALYZE`-Teil solch einer Graphbeschreibung gibt vor, welche Layer-Tier-Intersections für den resultierenden Graphen VKG_{v_i} in Betracht kommen. So werden

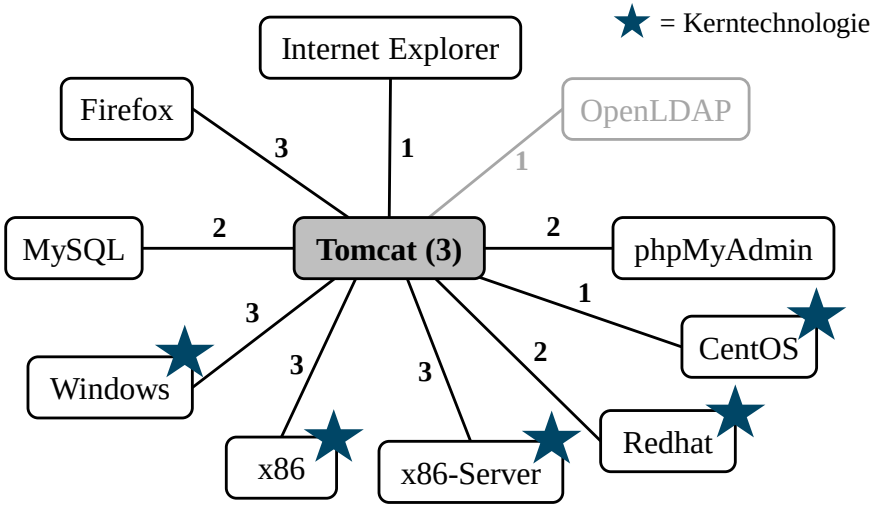


Abbildung 4.6: VKG_{Tomcat} für das fortlaufende Beispiel

alle Knoten aus dem Graphen entfernt, die keine Komponenten in den beschriebenen Intersections repräsentieren. Für unseren beispielhaften Teilgraphen VKG_{Tomcat} würde also der Knoten v_5 = `OpenLDAP` entfernt werden, da er sich auf dem Layer `Applikation` befindet. Im `BUILD`-Teil der Graphbeschreibung wird anhand der Anzahl oder Nutzungshäufigkeit von Komponenten bestimmt, wie viele Elemente im Graphen berücksichtigt werden sollen. So sind alle Knoten, welche diese Bedingung nicht erfüllen, aus dem Graphen zu entfernen. Im gezeigten Beispiel sind alle Komponenten mit einer Häufigkeit $H_{ko_a} \leq 10\%$ nicht für den Graphen VKG_{Tomcat} vorgesehen. Da diesen Wert aber alle betrachteten Elemente erfüllen, wird kein weiterer Knoten aus dem Graphen eliminiert. Basierend auf der Beschreibung im `CORE`-Teil, werden die verbleibenden Knoten im Graphen als Kerntechnologie identifiziert, wenn sie eine Komponente repräsentieren, die einer der definierten Kategorien entspricht. So werden im Beispielgraph VKG_{Tomcat} die Knoten v_6 = `x86`, v_7 = `x86-Server`, v_8 = `Windows`, v_9 = `Redhat` und v_{10} = `CentOS` als Kerntechnologie markiert. Die Abbildung 4.6 zeigt den generierten Beispielgraphen VKG_{Tomcat} für den Kandidaten `Tomcat`, der in insgesamt drei TAPs verbaut ist. Der ausgegraute Knoten v_5 = `OpenLDAP` stellt dabei ein eliminiertes Element dar.

4.2.3 Vergleich von variantenspezifischen Komponentengraphen

Durch den Vergleich von generierten VKGs für spezifische Restrukturierungspotentiale kann die technische Machbarkeit solcher Restrukturierungen abgeschätzt und bewertet werden. Da in diesem Zusammenhang lediglich Ersetzungspotentiale `RPi (A ERSETZT B)` über zwei Kandidaten (`A` und `B`) für einen Vergleich verfügen, werden Entfernungspotentiale `RPi (ENTFERNUNG C)` in dieser Phase nicht berücksichtigt. Allerdings werden die identifizierten VKGs für solche Entfernungspotentiale

im Rahmen der weiteren Analyse bewertet (vgl. Kapitel 5). Für Ersetzungspotentiale hingegen muss analysiert werden, ob ein ersetzender Kandidat `A` interoperabel mit den Komponenten ist, welche mit dem zu ersetzenden Kandidaten `B` in den verschiedenen TAPs verbaut sind. Um die Austauschfähigkeit dieser beiden Kandidaten zu bewerten, muss die *Ähnlichkeit* ihrer jeweiligen technischen Bebauung analysiert werden. Zu diesem Zweck führen wir eine vergleichende Analyse der betroffenen VKGs durch. So lässt sich anhand der Ähnlichkeit der verglichenen VKGs zweier Kandidaten `A` und `B` abschätzen, wie gut die technische Machbarkeit eines spezifischen Ersetzungspotentials `RPi (A ERSETZT B)` ist. Hierfür machen wir die folgende Annahme:

- Je höher die Ähnlichkeit der VKGs von zwei spezifischen Kandidaten, desto höher ist die technische Machbarkeit des betrachteten Ersetzungspotentials.

Diese Annahme gilt, da eine hohe Übereinstimmung von zwei betrachteten VKGs dazu führt, dass beide Kandidaten mit der Mehrheit von betroffenen Komponenten betreibbar sind und dadurch das Risiko für technische Limitierungen gering ist.

Da nicht jedes Element eines VKGs die gleiche Bedeutsamkeit für die technische Machbarkeit von Ersetzungspotentialen hat, unterscheiden wir im Rahmen der Vergleichsanalyse zwischen einem *Core-VKG* ($VKG_{C,i}$) und einem *Support-VKG* ($VKG_{S,i}$), welche beide Teilgraphen von VKG_i sind. Der Core-Graph stellt den essentiellen Bestandteil der technischen Architektur eines betrachten Kandidatens dar und wird durch die Kerntechnologie-Elemente definiert, welche im `CORE`-Teil einer Graphbeschreibung durch Technologieexperten beschrieben worden sind. Im Gegensatz dazu werden alle anderen Knoten des VKG_i dem Support-Graph zugeordnet. Diese Elemente stellen zwar keine essentiellen, aber dennoch wichtige, ergänzende Elemente für die technische Architektur dar.

Für den in Abbildung 4.6 vorgestellten VKG_{Tomcat} unseres beispielhaften Ersetzungspotentials `RP1` ergeben sich somit die beiden Teilgraphen $VKG_{C,Tomcat} =$ (`Windows, x86, x86-Server, Redhat, CentOS`) sowie $VKG_{S,Tomcat} =$ (`MySQL, Firefox, Internet Explorer, phpMyAdmin`). Für den zweiten Kandidaten `WebSphere App. Server` können die zwei Teilgraphen $VKG_{C,WebSphere} =$ (`Windows, x86, x86-Server, Redhat`) und $VKG_{S,WebSphere} =$ (`IBM DB2, Firefox`) identifiziert werden.

Um die Ähnlichkeit von zwei VKGs bestimmen zu können, werden diese also zunächst in ihre beiden Teilgraphen $VKG_{C,i}$ und $VKG_{S,i}$ zerlegt. Sind im `CORE`-Teil einer vorliegenden Graphbeschreibung keine Kategorien für die Kerntechnologie-Elemente definiert worden, so werden für den Vergleich keine Teilgraphen gebildet, sondern direkt die Graphen VKG_1 und VKG_2 der beiden Kandidaten `A` und `B` betrachtet.

Zur Ermittlung der Ähnlichkeit solcher VKGs wird der *Jaccard Index* verwendet. Der Jaccard Index (J) ist für zwei Mengen A und B wie folgt definiert [RV96]:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

Der Jaccard Index kann angewendet werden, um die *Gleichheit* beziehungsweise *Ähnlichkeit* von zwei Mengen zu bestimmen. Dabei werden die beiden Mengen A und B als gleich oder identisch bezeichnet, wenn jedes Element $a \in A$ auch Element von B ist, also auch $a \in B$ gilt. Der Jaccard Index stellt die Anzahl aller gleichen Elemente in A und B ins Verhältnis zur Anzahl aller Elemente der Mengen A und B. So kann überprüft werden, wie hoch der Anteil gleicher Elemente in A und B ist. Liegt dieser Anteil bei 100%, so sind beide Mengen gleich. Gilt ansonsten $0\% < J < 100\%$, dann weisen die beiden Mengen eine Ähnlichkeit von J auf.

Zur Bestimmung des Jaccard Indexes für zwei VKGs werden diese als Mengen betrachtet. So beschreibt die Menge V_1 alle Knoten $v_i \in VKG_1$ und die Menge V_2 alle Knoten $v_j \in VKG_2$. Sind auch die entsprechende Teilgraphen VKG_C und VKG_S vorhanden, so wird zwischen den Jaccard Indizes J_C für die Core-VKGs und J_S für die Support-VKGs unterschieden. Hierfür wird $VKG_{C,1}$ beziehungsweise $VKG_{S,1}$ als jeweilige Menge V_1 und $VKG_{C,2}$ beziehungsweise $VKG_{S,2}$ als entsprechende Menge V_2 betrachtet.

Die Anwendung des Jaccard Index für die beiden Graphen VKG_{Tomcat} und $VKG_{WebSphere}$ aus dem exemplarischen Ersetzungspotential `RP1` ergibt demnach eine Ähnlichkeit von $J_C = 0,8$ für $VKG_{C,Tomcat}$ und $VKG_{C,WebSphere}$ sowie $J_S = 0,2$ für $VKG_{S,Tomcat}$ und $VKG_{S,WebSphere}$.

Diese Ähnlichkeitswerte liefern einen guten Überblick darüber, wie viele der Komponenten in den betrachteten VKGs bereits zusammen mit dem ersetzenden Kandidaten `A` verwendet werden. Dadurch wird eine Bewertung der technischen Machbarkeit von Ersetzungspotentialen `RPi (A ERSETZT B)` ermöglicht. Zwar kann die technische Machbarkeit damit nicht garantiert werden, dennoch unterstützt dieser Ansatz Experten dabei, eine gut fundierte Abschätzung durchzuführen. So wird ein Ersetzungspotential dann als technisch machbar bewertet, wenn die Ähnlichkeit der betrachteten Core-VKGs bei $J_C \geq 0,8$ (Schwellwert S_{J_C}) und die Ähnlichkeit der betroffenen Support-VKGs bei $J_S \geq 0,5$ (Schwellwert S_{J_S}) liegt. Sind keine Core-VKGs vorhanden, sollte die allgemeine Ähnlichkeit $J \geq 0,5$ (Schwellwert S_J) betragen, damit ein entsprechendes Ersetzungspotential als technisch machbar betrachtet wird (vgl. Design-Entscheidung 4.2).

Um die Bedeutsamkeit eines ermittelten Ähnlichkeitswertes J in Bezug auf die technische Machbarkeit des zugrunde liegenden Ersetzungspotentials für Experten sichtbar zu machen, wird ein entsprechender *Confidence*-Wert co_J zugeordnet. Dies entspricht dem gleichen Vorgehen, welches auch für Restrukturierungspotentiale verwendet wird (vgl. Unterabschnitt 4.1.3). Hierdurch lässt sich für Experten leicht

erkennen, ob die Ähnlichkeitswerte von Ersetzungspotentialen für oder gegen eine technische Machbarkeit sprechen, ohne konkrete Ähnlichkeitswerte interpretieren zu müssen. So werden identifizierte Ähnlichkeitswerte J, J_C und J_S, welche ihren jeweiligen Schwellwert übersteigen mit $co_J = stark$ und jene Ähnlichkeitswerte, welche unter ihrem Schwellwert bleiben mit $co_J = schwach$ bewertet.

Design-Entscheidung 4.2: Schwellwerte S_{J_C}, S_{J_S} und S_J

Die vorgeschlagenen Schwellwerte S_{J_C}, S_{J_S} und S_J haben sich in Gesprächen mit den Experten unseres Industriepartners und durch Auswertung der vorhandenen Daten ergeben. Diese Werte können jedoch an individuelle Präferenzen angepasst werden. Allerdings empfehlen wir, den Schwellwert S_{J_C} höher zu wählen als den Wert für S_{J_S}, da die Elemente eines Core-VKGs essentielle Komponenten einer technischen Architektur darstellen und eine mögliche Ersetzung umso schwieriger wird, je geringer J_C ist. Hingegen ist eine Ersetzung potentiell auch bei niedrigem J_S möglich, da hiervon keine essentiellen Komponenten betroffen sind. Wird nur der Schwellwert S_J betrachtet, so sollte ein mittlerer Wert zwischen 0 und 1 gewählt werden, sodass in der nächsten Phase (vgl. Abschnitt 4.3) geeignete Restrukturierungsempfehlungen abgeleitet werden können. Diese lassen sich anschließend durch Experten mit ihrem technischen Domänenwissen überprüfen.

Für die beiden Ähnlichkeitswerte $J_C = 0,8$ und $J_S = 0,2$ aus dem Vergleich unserer beispielhaften Graphen VKG_{Tomcat} und $VKG_{WebSphere}$ ergeben sich bei Berücksichtigung der definierten Schwellwerte S_{J_C} und S_{J_S} die beiden Confidence-Werte $co_{J_C} = stark$ und $co_{J_S} = schwach$.

4.2.4 Identifizierung technischer Abhängigkeiten

Auf Basis der Ähnlichkeit von zwei VKGs kann die technischen Machbarkeit von potentiellen Ersetzungen zwar gut bewertet werden, jedoch bergen diejenigen Komponenten, die nicht in beiden VKGs enthalten sind, das Restrisiko einer technischen Abhängigkeit vom jeweiligen Kandidaten, was zu Inkompatibilitäten bei der Ersetzung führen kann.

Da jedoch für die Bestimmung solcher technischen Abhängigkeiten lediglich die Ist-Architekturen der vorhandenen TAPs zur Verfügung stehen, lässt sich nicht zweifelsfrei erkennen, ob sich zwei Komponenten tatsächlich technisch bedingen oder ob diese immer zusammen als beabsichtigter Unternehmensstandard verbaut sind. Da hierfür zusätzliche Informationen benötigt werden, können lediglich *potentielle technische Abhängigkeiten* und damit auch nur *potentielle Inkompatibilitäten* automatisiert bestimmt werden. Diese müssen anschließend durch Experten mit ihrem Domänenwissen verifiziert werden.

Definition 4.5: Potentielle technische Abhängigkeit

Im Kontext dieser Arbeit wird eine Komponente B als *potentiell technisch abhängig* von einer Komponente A betrachtet ($A \rightarrow B$), wenn (1) die Komponente A immer (in allen analysierten TAPs) zusammen mit der Komponente B verbaut ist und (2) B niemals mit einer anderen Komponente, welche die gleiche Kategorie wie A aufweist, verwendet wird.

Beispielsweise wird ein `Java Fat Client` im Normalfall immer zusammen mit der Komponente `Java Runtime Environment (JRE)` verbaut, sodass sich daraus die potentielle technische Abhängigkeit `JRE` $\rightarrow$ `Java Fat Client` ergibt. Wird nun versucht, solch eine Komponente durch eine dritte zu ersetzen, obwohl eine technische Abhängigkeit zu einer anderen Komponente wahrscheinlich ist, so kann dies zu einer *Inkompatibilität* führen.

Definition 4.6: Potentielle Inkompatibilität

Eine Komponente B wird als *potentiell inkompatibel* zu einer Komponente C bezeichnet, wenn die Komponente B abhängig von einer Komponente A ist, also $A \rightarrow B$ gilt, und die Komponenten C und A von der gleichen Kategorie sind.

Existiert also beispielsweise ein Ersetzungspotential `RP3 (.NET ERSETZT JRE)`, welches die Komponente `JRE` durch die Komponente `.Net` ersetzt, hat dies höchstwahrscheinlich zur Folge, dass die Applikation auf dem `Java FAT Client` nicht mehr funktioniert, da dieser technisch vom `JRE` abhängig ist. Somit wäre die Komponente `.Net` inkompatibel zur Komponente `Java FAT Client`, was in der folgenden Abbildung 4.7 dargestellt ist.

Potentielle technische Abhängigkeiten können sowohl für Ersetzungs- als auch für Entfernungspotentiale identifiziert werden. Diese führen allerdings erst dann zu potentiellen Inkompatibilitäten, wenn es ein entsprechendes Ersetzungspotential `RPi`

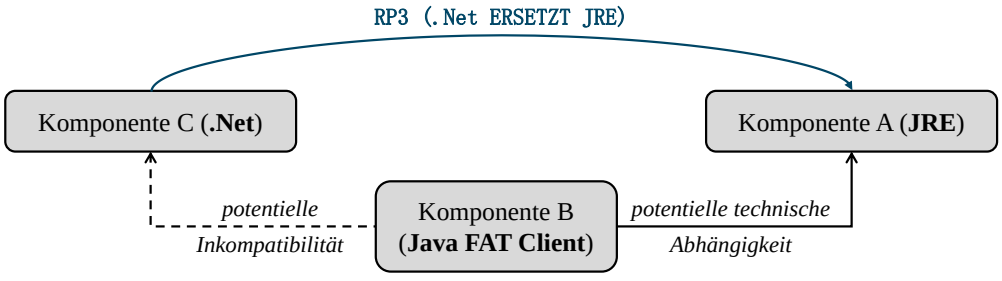


Abbildung 4.7: Technische Abhängigkeit und Inkompatibilität von Komponenten

gibt, welches auf den Austausch einer Komponente mit einer potentiellen technischen Abhängigkeit (z.B. `JRE`) abzielt. Um also potentielle Inkompatibilitäten zu bestimmen, sind zuerst die potentiellen technischen Abhängigkeiten zu identifizieren.

Zur Bestimmung von potentiellen technischen Abhängigkeiten zwischen einem Kandidaten eines Restrukturierungspotentials und den Komponenten, die mit diesem verbaut sind, wird der entsprechende VKG des jeweiligen Kandidaten analysiert. Dies erfolgt sowohl für Ersetzungs- als auch für Entfernungspotentiale. Allerdings wird für jedes Ersetzungspotential `RPi (C ERSETZT A)` nur der VKG_A betrachtet, da die vorgeschlagene Ersetzung nur Auswirkungen auf die Elemente haben kann, die mit der Komponente A verbaut sind. Dabei ist es für die Analyse wichtig, alle verbauten Elemente zu berücksichtigen, sodass auch Abhängigkeiten für solche Komponenten bestimmt werden können, die aufgrund einer spezifizierten Graphbeschreibung aus einem VKG_A eliminiert worden sind. Daher wird für die Identifizierung von technischen Abhängigkeiten der ursprüngliche VKG_A, also der aus dem Gesamtgraph G erstellte Teilgraph des betroffenen Kandidatens A ohne Berücksichtigung einer definierten Graphbeschreibung, betrachtet. Um nun zu bestimmen, ob eine Komponente B von einem Kandidaten A abhängig ist, wird zuerst deren gemeinsames *Vorkommen* betrachtet und anschließend die weitere *Verwendung* der Komponente B in TAPs ohne Kandidat A analysiert.

Bei der Betrachtung des Vorkommens von A wird untersucht, ob die Komponente B immer dann Verwendung findet, wenn auch der Kandidat A verbaut ist. Dazu wird das Gewicht f der Kante $e = \{A, B, f\} \in E(VKG_A)$ mit der Anzahl n aller TAPs verglichen, in denen der Kandidat A verbaut ist. Stimmen diese Werte überein (f=n), so lässt sich feststellen, dass der Kandidat A immer zusammen mit der Komponente B verbaut wird. Dieses gemeinsame Vorkommen in allen TAPs des Kandidaten A erfüllt das erste Kriterium für die technische Abhängigkeit $A \rightarrow B$ (vgl. Definition.4.5), sodass die Komponente B vorläufig als abhängig von A betrachtet wird.

Dieses Vorgehen kann beispielhaft anhand des VKG_{Tomcat} in Abbildung 4.8 demonstriert werden. Der hier betrachtete Kandidat `Tomcat` ist insgesamt in n=*3* TAPs implementiert und wird unter anderem immer mit der Komponente `x86-Server` zusammen verbaut, was sich aus dem Gewicht f=*3* an der Kante $e_1 = \{$`Tomcat`, `x86-Server`, 3$\} \in E(VKG_{Tomcat})$ und der daraus resultierenden Übereinstimmung f=n ergibt. So lässt sich vorläufig die technische Abhängigkeit `x86-Server` $\rightarrow$ `Tomcat` ableiten.

Um solch eine vorläufig identifizierte Abhängigkeit zu überprüfen, wird in einem nächsten Schritt die Verwendung der Komponente B in allen anderen TAPs untersucht, die nicht den Kandidaten A implementiert haben. Hierzu wird aus dem Gesamtgraph G ein eigenständiger Teilgraph VKG_B für die Komponente B erstellt, welcher alle Elemente beinhaltet, die zusammen mit der betrachteten Komponente B verbaut sind. Dieser VKG_B wird anschließend nach Elementen durchsucht, welche die gleiche Kategorie aufweisen, wie der betroffene Kandidat A. Lässt sich mindes-

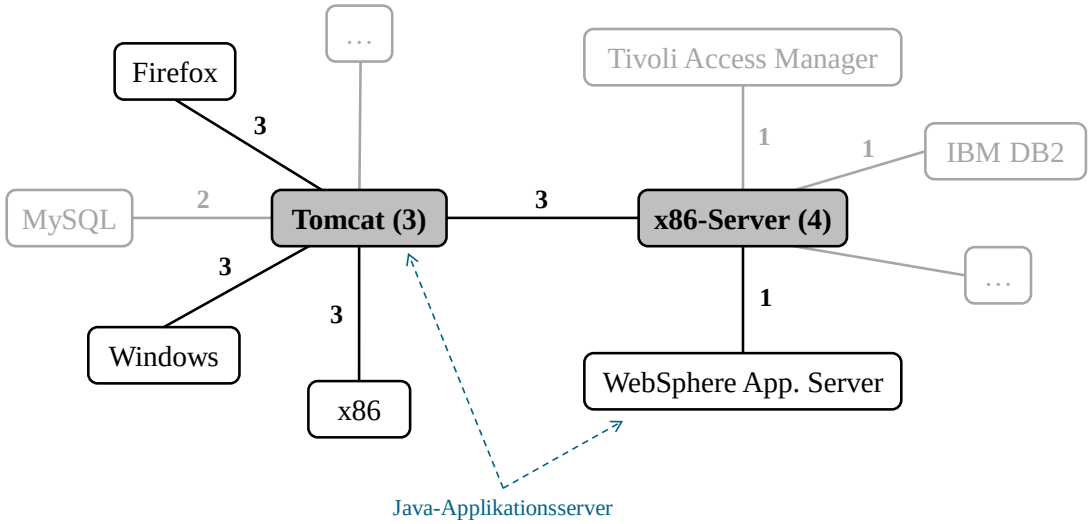


Abbildung 4.8: Ausschnitt aus den beiden VKGs für Tomcat und x86-Server

tens eine Komponente finden, welche diese Eigenschaft aufweist, so kann daraus abgeleitet werden, dass die Komponente B auch zusammen mit anderen funktional ähnlichen Komponenten betrieben werden kann. Dadurch wird das zweite Kriterium einer technische Abhängigkeit (vgl. Definition.4.5) nicht erfüllt, sodass eine vorläufige Abhängigkeit verworfen werden muss. Im umgekehrten Fall erfolgt bei Erfüllung dieses Kriteriums eine Bestätigung der ermittelten technischen Abhängigkeit $A \rightarrow B$. Für die vorläufig abhängige Komponente `x86-Server` aus dem beschriebenen Beispiel ergibt sich der in Abbildung 4.8 skizzierte $VKG_{x86-Server}$, welcher für die Kategorie *Java Applikationsserver* des betrachteten Kandidaten `Tomcat` eine weitere Komponente `WebSphere App. Server` beinhaltet. Insofern bestätigt sich die vorläufige Abhängigkeit `x86-Server` $\rightarrow$ `Tomcat` nicht und wird verworfen.

Nachdem alle potentiell technisch abhängigen Komponenten für ein Restrukturierungspotential identifiziert worden sind, werden diese dem jeweiligen Ersetzungs- oder Entfernungspotential hinzugefügt, um für spätere Analysen zur Verfügung zu stehen. Hieraus ergeben sich dann auch die entsprechenden potentiellen Inkompatibilitäten für die jeweiligen Ersetzungspotentiale.

Da sowohl die identifizierten Abhängigkeiten als auch die daraus resultierenden Inkompatibilitäten nur Potentiale darstellen, müssen diese anschließend durch Experten mit ihrem Domänenwissen verifiziert werden. Um dabei den manuellen Aufwand so gering wie möglich zu halten, werden zuerst potentiell technische Abhängigkeiten für alle Ersetzungs- und Entfernungspotentiale automatisiert bestimmt. Danach erfolgt die Ableitung konkreter Restrukturierungsempfehlungen ohne Berücksichtigung dieser identifizierten Abhängigkeiten (vgl. Abschnitt 4.3). Anschließend können Domänenexperten dann in der späteren Phase *Entscheidungsunterstützung* (vgl. Abschnitt 5.3) alle technischen Abhängigkeiten der für sie interessanten Restrukturie-

rungsempfehlungen einsehen und mit ihrem Wissen abgleichen. So entsteht der manuelle Aufwand für die Verifizierung von Abhängigkeiten nur bei solchen Restrukturierungsempfehlungen, die ohnehin von Experten näher betrachtet werden. Somit muss nicht jede einzelne Restrukturierungsempfehlung manuell überprüft werden. Wird dann im Rahmen solch einer manuellen Überprüfung eine konkrete Inkompatibilität festgestellt, können Domänenexperten die betroffene Restrukturierungsempfehlung einfach entfernen.

4.3 Ableitung konkreter Restrukturierungsempfehlungen

In der letzten Phase unseres Ansatzes steht die Ableitung von konkreten *Restrukturierungsempfehlungen* im Rahmen der identifizierten Kandidaten für unnötige Variabilität im Fokus.

Definition 4.7: Restrukturierungsempfehlung

Unter *Restrukturierungsempfehlung (RE)* verstehen wir eine empfohlene Umstrukturierung von TAPs zur Eliminierung von nicht erforderlichen technologischen Komponenten, welche zu unnötiger Variabilität führen.

Wie bei Restrukturierungspotentialen kann auch bei konkreten Restrukturierungsempfehlungen zwischen einer *Ersetzungsempfehlung* `REi (A ERSETZT B)` und einer *Entfernungsempfehlung* `REi (ENTFERNUNG C)` unterschieden werden. Dagegen bietet das Hinzufügen von Elementen keine Möglichkeit zur Reduzierung von Variabilität und wird deshalb nicht betrachtet. Wie in Abbildung 4.9 zu erkennen ist, können die beschriebenen Restrukturierungsempfehlungen auf Basis der Ergebnisse aus den beiden Vorphasen abgeleitet werden. Hierfür sind die identifizierten Restrukturie-

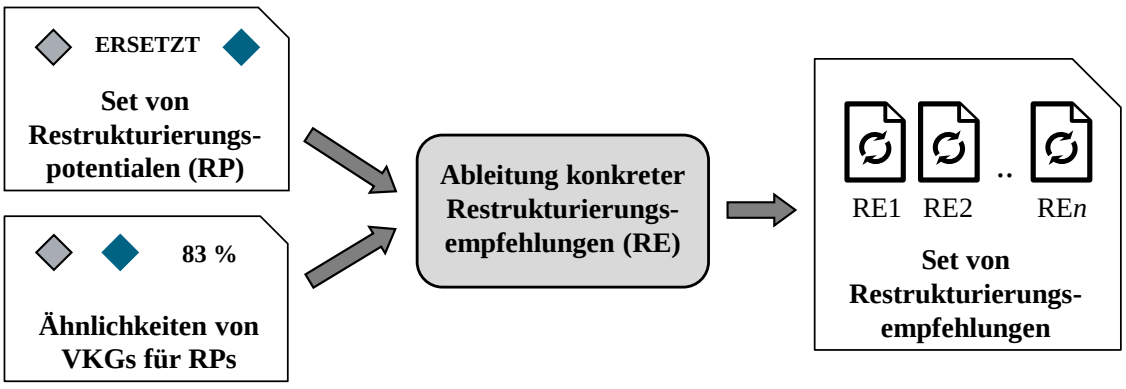


Abbildung 4.9: Workflow für die Ableitung von Restrukturierungsempfehlungen

rungspotentiale sowie die entsprechenden Ähnlichkeitswerte ihrer VKGs zu berücksichtigen.

Solche Restrukturierungsempfehlungen unterstützen Experten dabei, nicht benötigte Komponenten zu identifizieren und zu eliminieren. Daher erfolgt die Ableitung konkreter Restrukturierungsempfehlungen nur für Kandidaten, welche tatsächlich zu unnötiger Variabilität führen. Hierfür muss ein Kandidat sowohl aus businessorientierter als auch aus technischer Sicht als nicht erforderlich gelten. Dies kann auf Basis der entsprechenden Confidence-Werte aus der regelbasierten Businessanalyse und der graphbasierten Technologieanalyse bestimmt werden. Wie nachfolgend beschrieben, werden die identifizierten Confidence-Werte eines Ersetzungs- oder Entfernungspotentials zur Ableitung einer konkreten Ersetzungs- oder Entfernungsempfehlung herangezogen.

Ableitung von konkreten Ersetzungsempfehlungen

Die identifizierten Ersetzungspotentiale aus dem Set aller Restrukturierungspotentiale bilden die Grundlage für die Ableitung von Ersetzungsempfehlungen. Eine konkrete Ersetzungsempfehlung `REi (A ERSETZT B)` für einen Kandidaten `B` wird aus einem betrachteten Ersetzungspotential `RPi (A ERSETZT B)` abgeleitet, wenn die technische Machbarkeit des entsprechenden Restrukturierungspotentials erkannt wurde. Da dies von den jeweiligen Confidence-Werten co_J beziehungsweise co_{J_C} und co_{J_S} abhängt, wird bei der Ableitung von Ersetzungsempfehlungen unterschieden, ob die entsprechenden Core-VKGs für das jeweilige Ersetzungspotential `RPi (A ERSETZT B)` bestimmt worden sind oder nicht.

Die folgende Tabelle 4.3 stellt die Grundlage für die Ableitung von konkreten Ersetzungsempfehlungen ohne analysierten Core-VKG auf Basis der Confidence-Werte co_{RP} und co_J dar. Dabei wird die Erforderlichkeit des Kandidaten `B` eines spezifischen Ersetzungspotentials `RPi (A ERSETZT B)` mit co_{RP} aus einer businessorientierten Sicht und mit co_J aus einer technischen Sicht bewertet. Ein Kandidat `B` kann aus dieser technischen Perspektive als nicht erforderlich betrachtet werden, wenn $co_J = stark$ gilt, da hierdurch die technische Machbarkeit für die potentielle Ersetzung `RPi` gegeben ist und somit keine zweite technischen Lösung erforderlich

co_{RP}	co_J	Ableitung REi	co_{RE}
stark	stark	ja	stark
stark	schwach	nein	–
schwach	stark	ja	mittel(RP)
schwach	schwach	nein	–

co = Confidence RP = Restrukturierungspotential RE = Restrukturierungsempfehlung

Tabelle 4.3: Ableitung von Ersetzungsempfehlungen ohne analysiertem Core-VKG

ist. Aus businessorientierter Sicht kann der Kandidat `B` als nicht erforderlich angesehen werden, wenn $co_{RP} = stark$ für `RP`i gilt, da der Kandidaten `A` die Geschäftsanforderungen und -ziele somit besser unterstützt als Kandidat `B`. In dem Fall, dass $co_{RP} = stark$ und $co_J = stark$ gilt, wird für `RPi` eine neue Ersetzungsempfehlung `REi (A ERSETZT B)` erstellt, welche mit dem Confidence-Wert $co_{RE} = stark$ versehen wird, um zu signalisieren, dass der betroffene Kandidat `B` sowohl aus businessorientierter als auch aus technischer Sicht als nicht erforderlich betrachtet wird. Ist hingegen $co_{RP} = schwach$, so erfüllen beide Kandidaten aus `RPi` die beschriebenen Anforderungen und Ziele gleich gut, sodass ein Experte mit seinem zusätzlichen Domänenwissen bewerten muss, ob eine Ersetzung von `B` durch `A` wirklich sinnhaft ist. Dies wird für Experten dadurch ersichtlich, dass bei vorliegendem $co_{RP} = schwach$ und $co_J = stark$ eine neue Ersetzungsempfehlung `REi` aus `RPi` abgeleitet wird, welche den Confidence-Wert $co_{RE} = mittel(RP)$ zugewiesen bekommt. Dies signalisiert, dass die Bedeutsamtkeit der erstellten `REi` nur mittelmäßig aufgrund des schwachen co_{RP} ist. In den anderen beiden Fällen, die in der Tabelle 4.3 aufgeführt sind, werden keine Ersetzungsempfehlungen abgeleitet, da die technische Machbarkeit nicht gegeben ist ($co_J = schwach$). So werden die betroffenen Ersetzungspotentiale `RPi` verworfen.

In Tabelle 4.4 stellen wir die Grundlage für die Ableitung von konkreten Ersetzungsempfehlungen vor, die bei Vorhandensein von analysierten Core-VKGs greift. Hierbei werden zur Bewertung der Erforderlichkeit eines Kandidaten `B` aus einem spezifischen Ersetzungspotential `RPi (A ERSETZT B)` die Confidence-Werte co_{RP}, co_{J_C} und co_{J_S} herangezogen. Dabei wird co_{RP} zur Bewertung des Kandidaten `B` aus businessorientierter Sicht genauso betrachtet, wie bei der Ableitung von Ersetzungsempfehlungen ohne Core-VKG. Dagegen findet die Bewertung aus technischer Sicht anhand der beiden Confidence-Werte co_{J_C} und co_{J_S} statt, wobei co_{J_C} dabei im Vordergrund steht,

$\mathbf{co_{RP}}$	$\mathbf{co_{J_C}}$	$\mathbf{co_{J_S}}$	**Ableitung REi**	$\mathbf{co_{RE}}$
stark	stark	stark	ja	stark
stark	stark	schwach	ja	mittel(J_S)
stark	schwach	stark	nein	–
stark	schwach	schwach	nein	–
schwach	stark	stark	ja	mittel(RP)
schwach	stark	schwach	ja	mittel
schwach	schwach	stark	nein	–
schwach	schwach	schwach	nein	–

co = Confidence J = Ähnlichkeit (Jaccard) RE = Restrukturierungsempfehlung

Tabelle 4.4: Ableitung von Ersetzungsempfehlungen mit analysiertem Core-VKG

da hiermit die essentiellen Kernelemente einer technischen Architektur berücksichtigt werden. So wird eine konkrete Ersetzungsempfehlung `REi` für ein spezifisches Ersetzungspotential `RPi` erstellt, wenn $co_{RP} = stark$ und $co_{J_C} = stark$ gilt. Ist dabei ebenfalls $co_{J_S} = stark$, so bekommt die abgeleitet Empfehlung den Confidence-Wert $co_{RE} = stark$ zugewiesen. Bei $co_{J_S} = schwach$ wird der Wert $co_{RE} = mittel(J_S)$ zugewiesen, um Experten zu signalisieren, dass solch eine Ersetzungsempfehlung Kandidaten berücksichtigt, deren Support-VKGs lediglich eine schwache Ähnlichkeit aufweisen. Darüber hinaus werden zudem konkrete Ersetzungsempfehlungen abgeleitet, wenn $co_{RP} = schwach$ und $co_{J_C} = stark$ gilt. Hierdurch können auch für Kandidaten, welche aus businessorientierter Sicht als gleichwertig identifiziert worden sind, entsprechende Ersetzungen ermöglicht werden. Da auch solche Ersetzungsempfehlungen durch Experten mit ihrem Domänenwissen bewertet werden sollten, bekommen sie einen aussagefähigen Confidence-Wert zugeordnet. Gilt in diesem Zusammenhang $co_{J_S} = stark$, weist der Wert $co_{RE} = mittel(RP)$ auf $co_{RP} = schwach$ hin. In dem Fall, dass $co_{J_S} = schwach$ ist, bekommt die abgeleitete Ersetzungsempfehlung `REi` den Wert $co_{RE} = mittel$ zugewiesen, der sowohl auf $co_{RP} = schwach$ als auch auf $co_{J_S} = schwach$ hindeutet.

Alle anderen Fälle, die in der Tabelle 4.4 betrachtet werden, führen nicht zur Ableitung einer konkreten Ersetzungsempfehlung, da die technische Machbarkeit für entsprechende Ersetzungspotentiale nicht gegeben ist ($co_{J_C} = schwach$), sodass diese verworfen werden.

Für das Ersetzungspotential `RP1 (Tomcat ERSETZT WebSphere App. Server)` aus unserem fortlaufenden Beispiel wurden die jeweiligen Core-*VGKs* bestimmt und damit auch die entsprechenden Confidence-Werte $co_{RP} = stark$, $co_{J_C} = stark$ und $co_{J_S} = schwach$ ermittelt. Auf Basis dieser Werte lässt sich so mit Hilfe der Tabelle 4.4 die konkrete Ersetzungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` mit dem Confidence-Wert $co_{RP} = mittel(J_S)$ ableiten.

Ableitung von konkreten Entfernungsempfehlungen

Im Gegensatz zu Ersetzungspotentialen kann für identifizierte Entfernungspotentiale `RPi (ENTFERNUNG C)` kein Vergleich von Graphen stattfinden, da kein zweiter VKG_2 eines entsprechenden Kandidatens vorhanden ist, mit dem der VKG_1 des betroffenen Kandidaten `C` aus dem Ersetzungspotential verglichen werden kann. Um dennoch Experten bei der Einschätzung der technischen Machbarkeit solcher Entfernungspotentiale zu unterstützen, werden für diese sowohl potentielle technische Abhängigkeiten (vgl. Unterabschnitt 4.2.4) als auch potentielle Adaptionsabhängigkeiten (vgl. Unterabschnitt 5.1.2) identifiziert.

Der Hauptzweck solch einer Entfernungsempfehlung ist es, kaum verwendete Einzellösungen zu identifizieren und Experten auf diese aufmerksam zu machen. Durch die Ableitung konkreter Entfernungsempfehlungen können Experten solche Insellösungen leicht erkennen und auf Basis des 150%-Modells feststellen, dass

es im Rahmen der analysierter TAPs keine alternativ eingesetzten Komponenten gibt. So helfen konkrete Entfernungsempfehlungen Experten dabei, identifizierte Einzellösungen durch ihr Domänenwissen mit weiteren Komponenten außerhalb der betrachteten TAPs zu vergleichen und entsprechende Restrukturierungspotentiale manuell abzuleiten.

Die Ableitung einer konkreten Entfernungsempfehlung `REi` kann also nur aus businessorientierter Sicht erfolgen, indem der entsprechende Confidence-Wert co_{RP} eines spezifischen Entfernungspotentials `RPi` herangezogen wird. Da solch ein Entfernungspotential allerdings immer mit $co_{RP} = schwach$ bewertet ist, kann auch aus businessorientierter Sicht keine differenzierte Betrachtung erfolgen. Aus diesem Grund wird für jedes Entfernungspotential `RPi` eine entsprechende Entfernungsempfehlung `REi` abgeleitet. Solche Entfernungsempfehlungen müssen anschließend von Experten mit ihrem zusätzlichen Domänenwissen sowohl aus businessorientierter als auch aus technologieorientierter Perspektive bewertet werden. Hierzu können unter anderem die identifizierten Abhängigkeiten herangezogen werden. Um Experten die Erforderlichkeit solch einer manuellen Nachprüfung zu signalisieren, bekommen konkrete Entfernungsempfehlungen stets den Confidence-Wert $co_{RE} = schwach$ zugewiesen.

Für unser beispielhaftes Entfernungspotential `RP2 (ENTFERNUNG phpMyAdmin)` wird demnach `RE2 (ENTFERNUNG phpMyAdmin)` als konkrete Entfernungsempfehlung mit dem Confidence-Wert $co_{RE} = schwach$ abgeleitet.

Mit Hilfe des vorgestellten Ansatz ist es nun ohne großen manuellen Aufwand möglich, eine beliebige Anzahl von TAPs gleichzeitig zu analysieren, um deren Variabilität zu managen. Anhand von konkreten Restrukturierungsempfehlungen sind Experten jetzt in der Lage, unnötige Variabilität in einem Set von analysierten TAPs zu erkennen. So können sie sich auf die Bewertung von Restrukturierungsmaßnahmen fokussieren, um unnötige Variabilität in verwandten technischen Architekturen nachhaltig zu reduzieren.

4.4 Verwandte Arbeiten

Nachfolgend werden die verwandten Arbeiten vorgestellt, welche im Kontext dieses Kapitels betrachtet werden müssen. Dabei liegt der Fokus auf solchen Ansätzen, die zur Reduzierung der Variabilität in Technologiearchitekturen beitragen können. Da diese Fachlichkeit in der EA-Domäne geprägt ist, werden im Wesentlichen Arbeiten aus diesem Gebiet berücksichtigt.

Gruber et al. [GHK17, SHGZ17] stellen in diesem Zusammenhang einen guten Überblick über verfügbare Ansätze zum Management der IT-Komplexität vor. Nach den Autoren lassen sich diese in die Kategorien *Prozesse*, *Technologien*, *Messmethoden*, *Architekturmodelle*, *Vorgehensmodelle*, *Handlungsempfehlungen* und *Standardisierung* unterteilen. Für die Reduzierung von Variabilität in Technologiearchitekturen sind dabei die letzten drei Kategorien von besonderer Bedeutung. Während Vorge-

hensmodelle auf die Frage, *wie* Variabilität reduziert werden kann fokussieren, liefern Handlungsempfehlungen eine Antwort auf das *was*. Dabei kann die Standardisierung als spezielle Handlungsempfehlung verstanden werden [GHK17, SHGZ17], welche sich durch eine schriftlich fixierte Regel oder Richtlinie zum Zweck der Harmonisierung und Optimierung auszeichnet [DUA14]. Daher wird im folgenden auf verwandte Arbeiten aus den beiden Kategorien *Vorgehensmodelle* und *Handlungsempfehlungen* eingegangen.

Verwandte Arbeiten aus der Kategorie Vorgehensmodelle

Vorgehensmodelle schaffen ein Rahmenwerk, mit dessen Hilfe Maßnahmen zur Reduzierung von Variabilität in Technologiearchitekturen identifiziert und umgesetzt werden können.

Durst [Dur08] stellt in seiner Arbeit ein wertorientiertes Modell vor, welches die Konformität einer Architektur bewertet und die resultierenden Ergebnisse zur Entscheidung für die Implementierung solch einer Architektur zu Grunde legt. Basierend auf dem Vorgehensmodell *Variant Mode and Effects Analysis* können Architekturvarianten so analysiert, bewertet und ausgewählt werden, um deren Variantenvielfalt zu beherrschen und zu verringern.

Hanschke [Han10, Han11] präsentiert einen Prozess zur Standardisierung von Technologien, mit dem eine *Blueprint-Grafik* für die IT-Architektur erstellt werden kann. Diese beinhaltet technische Standards für Infrastrukturkomponenten und -plattformen sowie für Systeme und Schnittstellen. Durch den iterativ ausgelegten Prozess können solche Standards weiterentwickelt werden, sodass die Vielfalt und Heterogenität der betrachteten Artefakte reduziert werden kann und in einem geeignet Maß zu den Geschäftsanforderungen steht.

Keller [Kel12] stellt einen weiteren Ansatz zur Reduzierung von Heterogenität in IT-Architekturen vor. Dieser basiert auf einem 3-stufigen Managementprozess, welcher in der ersten Stufe die *Identifizierung der Heterogenität* fokussiert und in der zweiten Stufe eine *Wirtschaftlichkeitsrechnung* für die Beseitigung der identifizierten Heterogenität vorsieht. Eine *Planung und Umsetzung* der Beseitigung erfolgt anschließend in der dritten Stufe.

Biedermann et al. [BLS11] verstehen die Vielfalt parallel eingesetzter Infrastruktur- und Softwaretechnologien als branchenübergreifendes IT-Strukturproblem und schlagen zu dessen Lösung ein Vorgehensmodell zur strategischen IT-Bebauungsplanung vor. Dieses besteht aus den fünf Schritten *Anforderungsanalyse*, *Definition der Zielarchitektur*, *Bewertung der Ist-Bebauung*, *Bestimmung der Ziel-Bebauung* sowie *Erstellung Roadmap und Migrationsplan*. Hierdurch ist eine fachlich-orientierte Transformation der IT-Architektur möglich, womit veränderte oder neue Geschäftsanforderungen berücksichtigt werden können, um die Heterogenität der IT-Landschaft zu managen.

Matthes et al. [BELM08, KHSM15, SM15] präsentieren die *EAM Pattern Language* und stellen dafür einen Katalog mit geeigneten Mustern zur Verfügung, welche unterschiedliche Tätigkeiten im Rahmen des EAMs beschreiben und dabei

verschiedene Gesichtspunkte betrachten. So werden beispielsweise Muster für *Stakeholder*, *Concerns*, *Methoden*, *Viewpoints* und *Informationsmodelle* berücksichtigt und miteinander in Beziehung gesetzt. Einer dieser Concerns ist beispielsweise die *Identifizierung von Konsolidierungspotentialen*, welcher in der dazugehörigen Studie als besonders relevant eingestuft wurde. Durch die Zuordnung von ausgewählten Mustern für Methoden und Viewpoints wird so ein Vorgehen beschrieben, wie dieser und andere Concerns mit entsprechenden Tätigkeiten umgesetzt werden können.

Die vorgestellten Ansätze liefern zwar geeignete Vorgehensmodelle für die Reduzierung von Variabilität in Technologiearchitekturen, jedoch beschreiben diese häufig nur die auszuführende Tätigkeiten und sind lediglich auf manuelle Handlungen ausgerichtet. Somit bleibt die wesentliche Herausforderung in gewachsenen IT-Landschaften für Experten weiter bestehen: eine große Anzahl von unterschiedlichen Technologiearchitekturen zu analysieren und zu bewerten sowie daraus geeignete Restrukturierungsmaßnahmen zur Reduzierung der unnötigen Variabilität abzuleiten. Dagegen bietet unser Ansatz geeignete (semi-)automatisierte Methoden an, welche Experten bei der Bewältigung dieser Herausforderung unterstützen. So ist es ihnen möglich, konkrete Restrukturierungsempfehlungen zur Reduzierung von unnötiger Variabilität in großen, gewachsenen Technologiearchitekturen mit wenig manuellem Aufwand zu generieren.

Verwandte Arbeiten aus der Kategorie Handlungsempfehlungen

Neben Vorgehensmodellen lassen sich in der Literatur ebenfalls Ansätze finden, welche verschiedene Handlungsempfehlungen für die Reduzierung von Variabilität in Technologiearchitekturen vorschlagen.

Schütz et al. [SWG13] haben in ihrer Arbeit ein Set von sieben Design Prinzipien aufgestellt, welche Experten dabei unterstützen, die Komplexität von IT-Architekturen zu managen. Hierzu zählen beispielsweise die Berücksichtigung der *Anzahl* und der *Heterogenität von Komponenten und ihren Beziehungen* sowie die Betrachtung der *Änderungsrate* und des *Abstraktionslevels* von IT-Architekturen.

Schatz et al. [SSJ14] präsentieren einen Ansatz zum Management von Komplexität, welcher im Rahmen der Handlungsfelder Umgang, Reduzierung, Vermeidung, Bepreisung und Generierung von Komplexität bei der Ableitung von entsprechenden Maßnahmen unterstützt. Im Hinblick auf die Reduzierung schlagen sie konkret den Abbau von Überkomplexität vor, beispielsweise durch die *Eliminierung von Varianten* und der *Reduzierung von Schnittstellen*.

Beese et al. [BKAV17] stellen in ihrer Arbeit einen Ansatz zur Steuerung von IT-Komplexität mit Hilfe von institutionellen Strukturen vor. So leiten sie entsprechende Handlungsempfehlungen aus den drei institutionellen Säulen *regulativ*, *normativ* und *kulturell-kognitiv* ab. Hier schlagen sie beispielsweise Regeln und Vorschriften als regulative Handlungsempfehlung vor, mit deren Hilfe heterogene Technologievarianten standardisiert werden können.

Hanschke [Han11] beschreibt ebenfalls Handlungsfelder, welche adressiert werden müssen, um die IT-Komplexität zu beherrschen. Darunter fallen beispielsweise die Beseitigung von Vielfalt und Heterogenität durch die *Einführung geeigneter Technologiestandards* sowie das Vereinfachen (Aufräumen) der IT-Landschaft, indem *unnötige oder redundante Systeme, Funktionen oder Technologievarianten eliminiert* werden.

Grebe et al. [GD13] präsentieren einen Ansatz, um unnötige IT-Komplexität, welche keinen Mehrwert liefert, im Rahmen von identifizierten Handlungsfeldern zu reduzieren. Hierzu zählen unter anderem die *Reduzierung der Vielfalt von Technologie-Mustern* auf Ebene der Infrastruktur sowie die szenariobasierte *Rationalisierung von Applikationen.*

Fauscette et al. [FP14] stellen in ihrer Arbeit den *IT Complexity Index* vor, welcher verschiedene Indikatoren für unterschiedliche Handlungsfelder zur Reduzierung von IT-Komplexität berücksichtigt. Zu diesen gehören beispielsweise die *Konsolidierung und Rationalisierung* von Applikationen, Systemen und Technologie-Komponenten (z.B. Datenbanken), die *Modernisierung* von Applikationen und ihren technologischen Komponenten sowie die *Vereinheitlichung* der Betriebsumgebungen, insbesondere auf der Infrastrukturebene, wie zum Beispiel bei dem Einsatz von Betriebssystemen.

Albayrak et al. [AG14] zeigen mit ihrem Ansatz Handlungsempfehlungen auf, welche die Standardisierung als Mittel zur Komplexitätsverminderung in den Vordergrund stellen. Hierzu unterscheiden sie die vier Kategorien *Einfache IT*, *Komplizierte IT*, *Relativ komplexe IT* sowie *Äußerst komplexe IT*, welche sie anhand der Vielfalt von Elementen und ihrer Änderungshäufigkeit fest machen. Zur Reduzierung dieser Komplexität schlagen sie neben der Verwendung von branchenüblichen Standards für Prozesse und Vorgehensmodelle vor allem die *Standardisierung von Hardware- und Softwarekomponenten* vor.

Akella et al. [ABR09] präsentieren ebenfalls Handlungsempfehlungen zur Reduzierung von IT-Komplexität, welche sich direkt auf die Reduzierung der Variabilität auswirken. So schlagen sie beispielsweise die *Konsolidierung von Datenbanken* und die Entwicklung eines integrierten Datenmodells, die *Standardisierung von Technologien* zur Verringerung von redundanten Versionen und nicht weiter unterstützten Komponenten sowie die *Konsolidierung von Softwaresystemen mit vergleichbarer Funktionalität* vor.

Obwohl die hier vorgestellten Ansätze sinnvolle Handlungsfelder für die Reduzierung der Komplexität und insbesondere der Variabilität von Technologiearchitekturen liefern, sind sie wenig geeignet, um konkrete Restrukturierungsmaßnahmen für unternehmensindividuelle Architekturen herzuleiten. Oftmals werden lediglich Schlagwörter mit kurzen Beschreibungen als Handlungsempfehlungen dargestellt, beispielsweise die Eliminierung von Varianten und die Reduzierung von Schnittstellen [SSJ14], die Konsolidierung und Rationalisierung von Technologie-Komponenten [FP14] oder die Standardisierung von Technologien [ABR09]. Solche Handlungsempfehlungen sind aber zu allgemein und bieten keine spezifischen

Methoden oder Konzepte, mit deren Hilfe konkrete Lösungen für eine Variabilitätsreduzierung erarbeitet werden können [GHK17]. Insofern stellen auch diese Ansätze kein automatisiertes Verfahren bereit, welches Experten bei der Identifizierung und Reduzierung unnötiger Variabilität von Technologiearchitekturen in gewachsenen IT-Landschaften unterstützt. Dagegen ist es mit unserem Ansatz möglich, konkrete Restrukturierungsempfehlungen für beliebige systemspezifische Technologiearchitekturen (sogenannte TAPs) abzuleiten, um deren unnötige Variabilität nachhaltig zu reduzieren. So schlägt unser Ansatz konkrete Ersetzungsempfehlungen `REi (A ERSETZT B)` sowie Entfernungsempfehlungen `REi (ENTFERNUNG C)` für alle betroffenen TAPs vor. Hierdurch erhalten Experten konkrete Maßnahmen zur Reduzierung der Variabilität in den analysierten TAPs.

Neben diesen Ansätzen, welche manuelle Handlungsempfehlungen zur Reduzierung von Variabilität präsentieren, existieren auch automatisierte Verfahren, die gegebene Technologiearchitekturen analysieren und entsprechende Architekturempfehlungen ausgeben können.

Binz et al. [BLNS12] schlagen in ihrer Arbeit einen Ansatz vor, der die komplexe Infrastruktur einer Technologiearchitektur mit Hilfe einer graphbasierten Struktur, dem sogenannten *Enterprise Tologogy Graph*, abbildet. Durch die Anwendung von spezifischen Algorithmen zur Verarbeitung dieses Topologie-Graphens können verschiedene Herausforderungen im Kontext des Managements von Unternehmensarchitekturen gelöst werden. So lassen sich automatisierte Techniken wie beispielsweise *Abstraktion*, *Aggregation* und *Segmentierung* dazu verwenden, gezielte Analysen für ausgewählte Bereiche der IT-Infrastruktur (z.B. für eine Organisationseinheit) durchzuführen und entsprechende Empfehlungen zum Management der Technologiearchitektur für diese Bereiche abzuleiten.

Buschle et al. [BJS13, JUB+13, Bus14] präsentieren einen Ansatz zur automatisierten Analyse von Modellen der Unternehmensarchitektur, welcher auch für Technologiearchitekturen eingesetzt werden kann. Diesen Ansatz bezeichnen sie als *Predictive Probabilistic Architecture Modeling Framework*, dessen Hauptmerkmal es ist, Objekte und Relationen von EA-Modellen zu analysieren, um Vorhersagen über bestimmte Aspekte (z.B. die Verfügbarkeit einer Applikation) treffen zu können. Hierfür werden Methoden zur Bewertung der Wahrscheinlichkeit angewendet, welche dabei auch Unsicherheiten (z.B. in Form von fehlenden Informationen) berücksichtigen. Auf dieser Basis lassen sich schließlich geeignete Architekturempfehlungen ableiten.

Bhat et al. [BSB+17] stellen in ihrer Arbeit ein System zur automatisierten Ableitungen von Empfehlungen für Softwarearchitekturen vor, welches auch technologische Komponenten einer systemspezifischen Technologiearchitektur berücksichtigt. Mit Hilfe dieses Systems können die architekturellen Komponenten in Architekturdokumenten automatisiert annotiert und anschließend alternative Architekturlösungen für diese annotierten Komponenten vorgeschlagen werden. Dabei greift das präsentierte System auf das Wissen zu, welches in öffentlich verfügbaren *Cross-Domain-Ontologies* (z.B. *DBpedia*) enthalten ist. So können alternative Lösungen für konkrete Architekturentscheidungen berücksichtigt werden.

Zwar bieten diese Ansätze automatisierte Methoden zur Analyse von Technologiearchitekturen und zur Ableitung von Architekturempfehlungen, jedoch sind sie nicht darauf ausgelegt, unnötige Variabilität in Technologiearchitekturen zu erkennen und entsprechende Restrukturierungsempfehlungen zu deren Reduzierung abzuleiten. Vielmehr stehen hierbei alternative Lösungsarchitekturen im Vordergrund, welche bewertet und mit Hilfe von Technologieempfehlungen realisiert werden können. Somit liefern auch diese Arbeiten keinen geeigneten Lösungsansatz, um Experten bei der Reduzierung von unnötiger Variabilität in gewachsenen Technologiearchitekturen zu unterstützen. Dagegen ist unser Ansatz in der Lage, konkrete Ersetzungs- oder Entfernungsempfehlungen für ein Set von systemspezifischen Technologiearchitekturen automatisiert zu generieren. So unterstützt unser Ansatz Experten mit konkreten Maßnahmen zur Reduzierung der unnötigen Variabilität in den analysierten TAPs.

4.5 Zusammenfassung

Die Identifizierung und Reduzierung von unnötiger Variabilität in Technologiearchitekturen ist für Experten bisher eine sehr herausfordernde und zeitaufwändige Tätigkeit, da es bislang keine unterstützenden Verfahren gibt, die durch geeignete Techniken und Automatismen den hohen manuellen Aufwand verringern. So ist die Analyse von technisch verwandten TAPs bisher schon für eine kleine Menge mühsam und für hunderte von unterschiedlichen Architekturen in gewachsenen IT-Landschaften händisch gar nicht mehr zu bewältigen. Zur Überwindung dieser Problematik haben wir in diesem Kapitel einen automatischen Ansatz für die Ableitung von Restrukturierungsempfehlungen zur Reduzierung von unnötiger Variabilität in analysierten TAPs vorgestellt. Mit dessen Hilfe sind Experten nun in der Lage, die Variabilität für eine beliebige Menge von TAPs in gewachsenen IT-Landschaften zu managen.

Hierfür wurde zuerst in Abschnitt 4.1 gezeigt, wie die identifizierten Kandidaten für unnötige Variabilität, basierend auf den Ergebnissen in Kapitel 3, aus einer businessorientierten Perspektive im Rahmen der *Regelbasierten Businessanalyse* bewertet werden. Dazu beschreiben Experten ihre Geschäftsanforderungen und -ziele in Form von Geschäftsregeln, welche für die automatisierte Bewertung von Kandidaten verwendet werden, um daraus Restrukturierungspotentiale abzuleiten.

Im nächsten Abschnitt 4.2 wurde dargestellt, wie solche Restrukturierungspotentiale anschließend im Rahmen der *Graphbasierten Technologieanalyse* aus einer technischen Perspektive analysiert und bewertet werden können. Zu diesem Zweck werden die betrachteten TAPs der jeweiligen Kandidaten als Graphen dargestellt und miteinander verglichen, um die technische Machbarkeit des entsprechenden Restrukturierungspotentials zu bestimmen.

Abschließend wurde im Abschnitt 4.3 vorgestellt, wie die Ergebnisse der beiden vorherigen Phasen in der dritten und letzten Phase unseres Ansatzes *Ableitung konkreter Restrukturierungsempfehlungen* weiter verarbeitet werden. In diesem Kontext

wird untersucht, ob ein Kandidat sowohl aus businessorientierter als auch aus technischer Sicht als nicht erforderlich angesehen werden kann und damit tatsächlich zu unnötiger Variabilität führt. Ist dies der Fall, wird für den betrachteten Kandidaten eine konkrete Restrukturierungsempfehlung abgeleitet, um diesen entweder mittels Ersetzung durch einen anderen Kandidaten oder mit Hilfe einer Entfernung eliminieren zu können. Solche Restrukturierungsempfehlungen unterstützen Experten beim Management der Variabilität von Technologiearchitekturen und der Entscheidungsfindung zur Umsetzung von konkreten Restrukturierungsmaßnahmen.

5 Entscheidungsunterstützung für Restrukturierungsempfehlungen

Dieses Kapitel basiert in wesentlichen Teilen auf der Veröffentlichung [WWSS18].

Um auf Basis der abgeleiteten Restrukturierungsempfehlungen (vgl. Kapitel 4) angemessene Entscheidungen für eine Restrukturierung der betrachteten TAPs treffen zu können, ist eine zusätzliche Bewertung notwendig, welche die Identifizierung geeigneter Restrukturierungsempfehlungen ermöglicht. Hierfür müssen die positiven sowie negativen Auswirkungen jeder einzelnen Restrukturierungsempfehlung analysiert werden. Dies umfasst beispielsweise die erreichbare Variabilitätsreduktion sowie erforderliche Anpassungen und damit verbundene Aufwände. Da solch eine Bewertung eine komplexe Aufgabe darstellt, die mit hohem manuellen Aufwand verbunden ist [GAE+16], führt dies zu zeitaufwändigen Tätigkeiten, welche für eine große Anzahl von Softwaresystemen durch Domänenexperten nicht geleistet werden können. Daher stellen wir in diesem Kapitel einen automatisierbaren Ansatz zur Bewertung von Restrukturierungsempfehlungen und zur anschließenden Unterstützung der Entscheidungsfindung vor. Dieser Ansatz besteht aus vier Phasen und ist in der folgenden Abbildung 5.1 schematisch dargestellt.

In der ersten Phase *Einzelbewertung* (vgl. Abschnitt 5.1) erfolgt eine Begutachtung jeder einzelnen Restrukturierungsempfehlung im Hinblick auf ihr Potential zur Va-

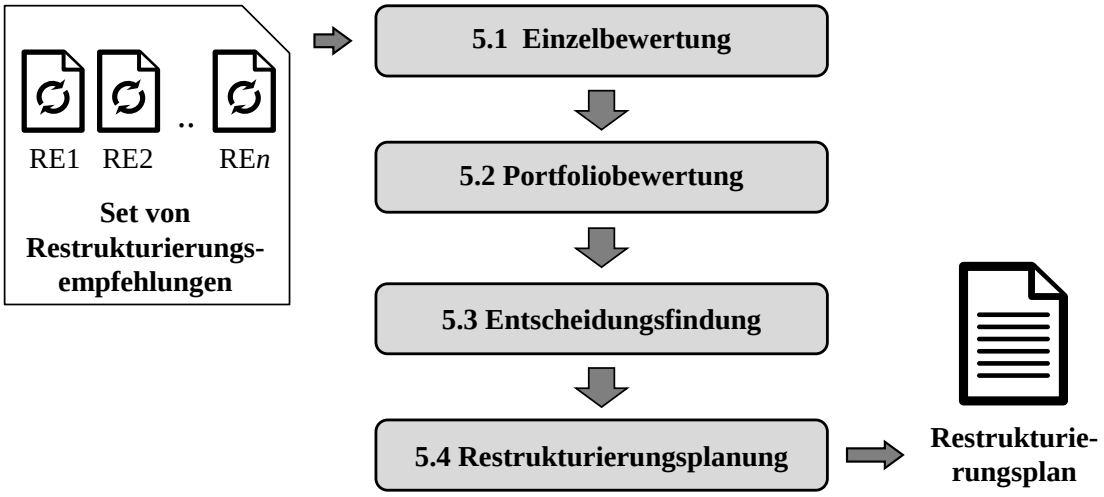


Abbildung 5.1: Workflow für die Entscheidungsunterstützung

riabilitätsreduzierung sowie den damit einhergehen Anpassungsbedarfen und dem dadurch entstehenden Aufwand. Dies stellt die *Phase E* unseres Gesamtansatzes dar (vgl. Abbildung 1.1). In der zweiten Phase *Portfoliobewertung* (vgl. Abschnitt 5.2) werden alle Restrukturierungsempfehlung im Vergleich zueinander analysiert, um diejenigen mit geeignetem Aufwand-Nutzen-Verhältnis zu identifizieren. Außerdem werden die Intersections mit hohem Restrukturierungspotential bestimmt. Die Ergebnisse beider Bewertungsphasen stehen dann in der dritten Phase *Entscheidungsfindung* (vgl. Abschnitt 5.3) zur Verfügung, welche Domänenexperten durch einen Entscheidungsprozess und die Möglichkeit der Simulation von Restrukturierungsempfehlungen unterstützt. Dabei realisieren die Portfoliobewertung und die Entscheidungsfindung die *Phase F* unseres Gesamtansatzes. In der letzten Phase *Restrukturierungsplanung* (vgl. Abschnitt 5.4) wird anschließend ein Restrukturierungsplan erzeugt, welcher alle Maßnahmen für jeden TAP auflistet, die zur Umsetzung der getroffenen Restrukturierungsentscheidungen erforderlich sind. Dies entspricht der *Phase G* unseres Gesamtansatzes.

5.1 Bewertung von einzelnen Restrukturierungsempfehlungen

Um Experten bei der Evaluierung von möglichen Restrukturierungen zu unterstützen, stellen wir in diesem Abschnitt einen Ansatz zur automatisierten Bewertung einzelner Restrukturierungsempfehlungen vor. Dieser ist in Abbildung 5.2 schematisch dargestellt und beinhaltet drei verschiedene Methoden, deren Einzelergebnisse an die jeweilige Restrukturierungsempfehlung angehangen werden, um für die Portfoliobewertung (vgl. Abschnitt 5.2) und die anschließende Entscheidungsfindung (vgl. Abschnitt 5.3) zur Verfügung zu stehen. Zusammen mit den bereits hinzugefügten tech-

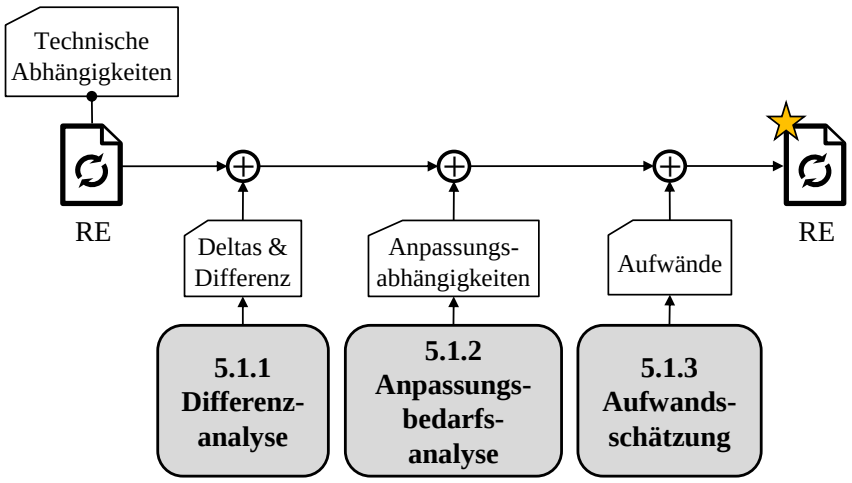


Abbildung 5.2: Workflow zur Bewertung von einzelnen Restrukturierungsempfehlungen

nischen Abhängigkeiten (vgl. Unterabschnitt 4.2.4) führen diese Ergebnisse zu einer bewerteten Restrukturierungsempfehlung (durch den Stern gekennzeichnet).

Im Rahmen der *Differenzanalyse* (vgl. Unterabschnitt 5.1.1) werden erforderliche Deltaoperationen für die spätere Simulation einer spezifischen Restrukturierungsempfehlung und basierend darauf ihr Potential zur Variabilitätsreduzierung bestimmt. In der *Anpassungsbedarfsanalyse* (vgl. Unterabschnitt 5.1.2) werden alle Komponenten identifiziert, die aufgrund der beabsichtigten Restrukturierung an die neue Architekturlösung angepasst werden müssen, damit das veränderte System weiterhin reibungslos funktioniert. Zuletzt werden dann in der Aufwandsschätzung (vgl. Unterabschnitt 5.1.3) die potentiellen Aufwände ermittelt, die mit dieser Restrukturierung und den erforderlichen Anpassungen verbunden sind.

5.1.1 Differenzanalyse

Zur Umsetzung einer Restrukturierungsempfehlung erfolgt eine Umstrukturierung der TAPs, die von der vorgeschlagenen Restrukturierung betroffen sind. Dies kann entweder dazu führen, dass bestimmte Komponenten eines TAPs auf Grundlage einer Ersetzungsempfehlung gegen funktional ähnliche Elemente ausgetauscht werden oder dass ausgewählte Komponenten auf Basis einer Entfernungsempfehlung aus einem TAP eliminiert werden. Dabei wird immer das Ziel verfolgt, eine Reduzierung der unnötigen Variabilität in den betrachten TAPs zu erreichen. Um eine Restrukturierungsempfehlung im Hinblick auf ihr Potential zur Reduzierung dieser Variabilität zu bewerten, führen wir eine *Differenzanalyse* durch, welche alle Modifikationen berücksichtigt, die durch diese Restrukturierungsempfehlung verursacht werden.

In Abbildung 5.3 sind die Zusammenhänge der Differenzanalyse graphisch dargestellt. Den Ausgangspunkt bildet dabei das generierte 150%-Modell (vgl. Kapitel 3), welches den *Ist-Zustand* der Variabilität für ein bestimmtes Set von TAPs darstellt.

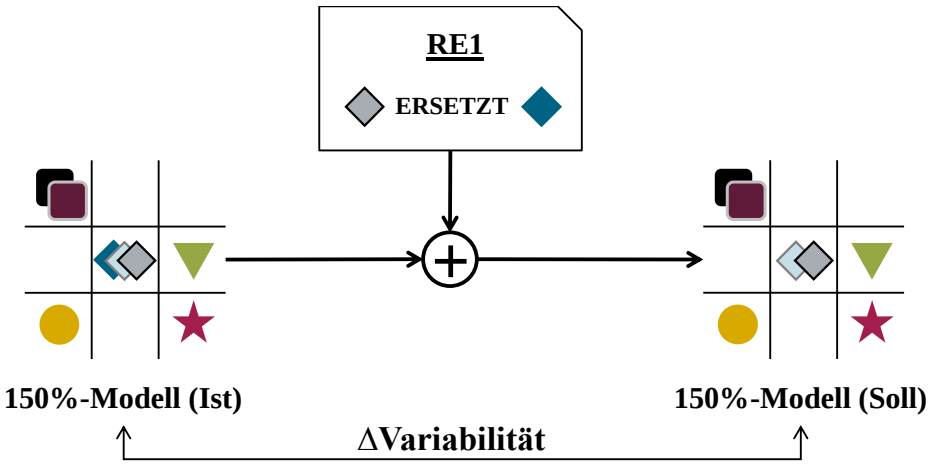


Abbildung 5.3: Zusammenhänge im Rahmen der Differenzanalyse

Durch die Anwendung einer Restrukturierungsempfehlung werden die betreffenden TAPs modifiziert, was zu einem veränderten 150%-Modell führt. Dieses veränderte Modell beschreibt den *Soll-Zustand* der Variabilität nach Umsetzung einer konkreten Restrukturierungsempfehlung. Die *Variabilitätsdifferenz (VD)*, die sich dann aus dem Vergleich des Ist-150%-Modells und des Soll-150%-Modells ergibt, stellt das Variabilitätsreduktionspotential dieser Restrukturierungsempfehlung für das betrachtete Set von TAPs dar.

Die sich aus der Umsetzung einer spezifischen Restrukturierungsempfehlung `REi` ergebene Variabilitätsdifferenz *VD(REi)* zwischen einem Ist-150%-Modell (M_1) und dem daraus resultierenden Soll-150%-Modell ($M_2(REi)$) lässt sich auf Basis der Gleichung 3.4 wie folgt bestimmen:

$$\begin{aligned} VD(REi) \;=\; \Delta Variabilität(M_1, M_2(REi)) \;=\; & \Delta N_{AG} \;+\; \Delta N_{RG} \\ & +\; \Delta N_A \;+\; \Delta N_O \;+\; \Delta N_{V_{RG}} \end{aligned} \tag{5.1}$$

Der in dieser Gleichung verwendete *Differenzoperator* Δ gibt dabei an, dass hier die Differenzwerte der jeweiligen Variablen betrachtet werden, die sich aus den beiden 150%-Modellen M_1 und $M_2(REi)$ ergeben. So gilt beispielsweise für die Differenz in der Anzahl von Alternativgruppen $\Delta N_{AG} = N_{AG}(M_1) - N_{AG}(M_2(REi))$.

Um die Variabilitätsdifferenz automatisiert bestimmen zu können, benötigen wir einen Ansatz, der ausgehend von einem konkreten Ist-150%-Modell die Generierung des resultierenden Soll-150%-Modells für eine spezifische Restrukturierungsempfehlung ermöglicht. Zu diesem Zweck verwenden wir den vorgestellten Transformationsmechanismus *Delta Modellierung* (vgl. Unterabschnitt 2.1.3), der eine automatische Transformation von Modellen auf Basis von definierten Deltaoperationen erlaubt. Mit Hilfe solcher Deltaoperationen kann ein Modell einer Basisvariante durch Hinzufügen, Entfernen oder Ändern von Modellelementen zu einem Modell der Soll-Variante transformiert werden. In unserem Ansatz repräsentiert das Ist-150%-Modell die entsprechende Basisvariante und eine konkrete Restrukturierungsempfehlung beschreibt die beabsichtigten Modifikationen für die Ableitung des entsprechenden Soll-150%-Modells.

Da eine Restrukturierungsempfehlung lediglich allgemeine Informationen über beabsichtigte Modifikationen enthält, muss diese zuerst in explizite Deltaoperationen übersetzt werden, bevor sie für die automatisierte Transformation eines Ist-150%-Modells in das entsprechende Soll-150%-Modell verwendet werden kann. Zu diesem Zweck haben wir eine domänenspezifische *Deltasprache* entwickelt, welche ein beschränktes Set von möglichen Deltaoperationen für 150%-Modelle von TAPs beschreibt. Mit Hilfe dieser Deltasprache können alle notwendigen Änderungsoperationen für eine spezifische Restrukturierungsempfehlung identifiziert werden, um damit das resultierende 150%-Modell zu erzeugen. Für die Entwicklung dieser Sprache haben wir *DeltaEcore*[1] verwendet, welches ein Tool-unterstütztes

[1] http://deltaecore.org/

```
deltaDialect
{
   deltaOperations:
      customOperation addContainingModelToModelElement(String
         value, ModelElement element);
      removeOperation removePhysicalElementFromLogicalElement(
         PhysicalElement value, LogicalElement [
         physicalElements] element);
      modifyOperation modifyVariabilityOfModelElement(
         Variability value, ModelElement [variability] element
         );

      [..]
}
```

Listing 5.1: Auszug aus dem entwickelten Deltadialekt

Framework zur Erstellung von domänenspezifischen Deltasprachen ist [SSA14]. Hierbei wird zwischen der *Common Base Delta Language* und dem *Delta Dialect* unterschieden. Ersteres stellt Basisfunktionalitäten für alle Deltasprachen bereit. Letzteres definiert hingegen sprachenindividuelle Deltaoperationen. So wurde für den hier beschriebenen Anwendungsfall ein Deltadialekt mit insgesamt 16 Deltaoperationen entwickelt, welcher alle Möglichkeiten zur Modifikation eines 150%-Modells von TAPs definiert.

Das Listing 5.1 zeigt einen Auszug aus dem entwickelten Deltadialekt. Eine vollständige Übersicht über alle definierten Deltaoperationen kann dem Anhang (vgl. Abschnitt A.2) entnommen werden. Der Auszug in Listing 5.1 zeigt drei Deltaoperationen, jeweils eine zum Hinzufügen, Entfernen und Ändern eines Elements des 150%-Modells. Dabei kann mit der ersten Deltaoperation `addContainingModelToModelElement(value, element)` das Hinzufügen eines weiteren konkreten Elementes zu einem spezifischen TAP erfolgen. Durch die nächste Deltaoperation `removePhysicalElementFromLogicalElement(value, element)` kann ein bestimmtes physisches Element von einem ausgewählten logischen Element entfernt werden. Mit Hilfe der dritten Deltaoperation `modifyVariabiliyOfModelElement(value, element)` kann zudem die Variabilitätsinformation von einem spezifischen Modellelement verändert werden.

Zur Umsetzung einer konkreten Restrukturierungsempfehlung werden alle hierfür notwendigen Deltaoperationen aus dem Set des entwickelten Deltadialektes ausgewählt, welche für die Modifikationen der betrachteten TAPs erforderlich sind. Dabei erfolgt die Identifikation der notwendigen Deltaoperationen auf Grundlage der Metamodell-Repräsentation von TAPs, um zu gewährleisten, dass alle entsprechenden Modellelemente berücksichtigt werden. So lässt sich sicherstellen, dass die jeweiligen TAPs auch nach ihrer Restrukturierung eine gültige Konfiguration darstellen.

```
deltaModul(RE1)
{
    removePhysicalElementFromLogicalElement('WebSphere App.
        Server 7.0', 'WebSphere App. Server');
    removeModelElementFromModel('WebSphere App. Server 7.0',
        'TAP3');
    removeLogicalElementFromEAModel('WebSphere App. Server',
        '150_model');
    addContainingModelToModelElement('TAP3', 'Tomcat 7');
    modifyVariabilityOfModelElement('mandatory', 'Tomcat');
}
```

Listing 5.2: Deltaoperationen für die Restrukturierungsempfehlung `RE1` aus dem fortlaufenden Beispiel

Die identifizierten Deltaoperationen werden anschließend zu einem Deltamodul zusammengefasst und der entsprechenden Restrukturierungsempfehlung angehangen, um für weitere Analysen zu Verfügung zu stehen. Für die exemplarische Restrukturierungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` aus unserem fortlaufenden Beispiel werden die fünf in Listing 5.2 gezeigten Deltaoperationen zur Transformation des 150%-Modells (vgl. Tabelle 3.12) benötigt.

Die erste Deltaoperation in Listing 5.2 beschreibt die Entfernung des physischen Elementes `WebSphere App. Server 7.0` vom logischen Element `WebSphere App. Server`. Die zweite Deltaoperation bestimmt die Entfernung desselben physischen Elementes aus dem `TAP3`. Dieser TAP ist der einzige aus unserem Beispiel, welcher dieses zu ersetzende Element beinhaltet (vgl. Tabelle 3.1). Die dritte Deltaoperation entfernt dann das logische Element `WebSphere App. Server` aus dem gesamten 150%-Modell, da dieses in keiner anderen Intersection und keinem anderen TAP mehr benötigt wird. Mit Hilfe der vierten Deltaoperation wird schließlich die neue Komponente `Tomcat 7` dem `TAP3` hinzugefügt. Abschließend wird dann mit der letzten Deltaoperation die Variabilität dieser Komponente so verändert, dass sie den Wert *verpflichtend* (`mandatory`) annimmt.

Auf Basis der generierten Deltaoperationen kann die *Variabilitätsdifferenz* für eine Restrukturierungsempfehlung ermitteln werden, welche sich aus der Transformation des Ist-150%-Modell für diese Restrukturierung ergeben würde. Zu diesem Zweck wird jede erstellte Deltaoperation daraufhin untersucht, welche Auswirkung sie auf die Variabilitätsbeziehungen innerhalb des Ist-150%-Modells hat. Beispielsweise würde eine Entfernung des `WebSphere App. Server 7.0` aus dem `TAP3` (Deltaoperation 2) zu einer Eliminierung der Alternative `A2` (vgl. Tabelle 3.12) führen. Darüber hinaus würde die Änderung der Variabilität des `Tomcat` (Deltaoperation 5) die Entfernung der Alternative `A1` und damit auch die Auflösung der Alternativgruppe `AG1` bedeuten. Daraus ergeben sich für `RE1` die beiden Werte $\Delta N_{AG} = -1$ und $\Delta N_A = -2$, was nach Gleichung 5.1 zu einer Variabilitätsdifferenz von $\Delta Variabilität(M_1, M_2(RE1)) = -3$ führt. Diese Differenz stellt das Poten-

tial zur Reduzierung der Variabilität dar, welche durch die betrachtete Restrukturierungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` für die analysierten TAPs (vgl. Tabelle 3.1) erreicht werden kann. Auch diese Informationen werden der entsprechenden Restrukturierungsempfehlung hinzugefügt, damit sie für weitere Analyse und der späteren Entscheidungsunterstützung von Domänenexperten (vgl. Abschnitt 5.3) zur Verfügung zu stehen.

5.1.2 Anpassungsbedarfsanalyse

Für die Stabilität eines Anwendungssystems ist es erforderlich, dass die einzelnen technologischen Komponenten des systemspezifischen TAPs und ihr Zusammenspiel einwandfrei funktionieren. Wird allerdings die Architektur des betrachteten Systems durch die Ersetzung einer Komponente verändert, kann dies dazu führen, dass bereits bestehende Architekturelemente nicht reibungslos mit der neu implementierten Komponente funktionieren und somit eine Anpassung der Altkomponenten (z.B. ihrer Schnittstellen oder Konfigurationsdateien) auf die technischen Anforderungen der Neukomponente erforderlich ist. Zudem kann auch die Entfernung einer Komponente aus solch einer Systemarchitektur dazu führen, dass übrige Elemente dieser Architektur angepasst werden müssen. Ist dies der Fall, so liegt eine *konkrete Anpassungsabhängigkeit* vor.

Definition 5.1: Konkrete Anpassungsabhängigkeit

Unter einer *konkreten Anpassungsabhängigkeit* wird eine direkte Abhängigkeit zwischen zwei Komponenten A und B (Komponente A → Komponente B) verstanden, welche dadurch begründet ist, dass eine Ersetzung oder Entfernung der Komponente A eine Anpassung der im gleichen TAP verbauten Komponente B erforderlich macht.

So kann beispielsweise die Umsetzung der exemplarischen Restrukturierungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` im `TAP3` dazu führen, dass die Datenbankkomponente `IBM DB2` für den neuen Applikationsserver angepasst werden muss. Daraus ergibt sich die konkrete Anpassungsabhängigkeit `WebSphere App. Server` → `IBM DB2` für dieses Beispiel. Um solche Anpassungsabhängigkeiten zu identifizieren, führen wir eine *Anpassungsbedarfsanalyse* durch.

Die Basis für die Anpassungsbedarfsanalyse bildet ein Abhängigkeitsmodell für *abstrakte Anpassungsabhängigkeiten*. Dieses Modell beschreibt allgemeingültige Anpassungsabhängigkeiten zwischen Elementen von sich gegenseitig beeinflussenden Intersections innerhalb eines TAPs. Solche abstrakten Anpassungsabhängigkeiten betrachten dabei zunächst nur die Intersections von TAPs und dienen als Grundlage für die Ableitung von konkreten Anpassungsabhängigkeiten, die zwischen den einzelnen Elementen, welche in den betrachteten Intersections verbaut sind, bestehen.

Definition 5.2: Abstrakte Anpassungsabhängigkeit

Eine *abstrakte Anpassungsabhängigkeit* beschreibt eine direkte Abhängigkeit zwischen zwei Intersections A und B (Intersection A → Intersection B), welche ausdrückt, dass eine Ersetzung oder Entfernung einer beliebigen Komponente in der Intersection A eine Anpassung der im gleichen TAP in Intersection B verbauten Komponenten erforderlich macht.

Solch eine abstrakte Anpassungsabhängigkeit beschreibt einen typischen Anpassungsbedarf zwischen Elementen zweier Intersections und kann beispielsweise die folgende sein: `Applikation-Server-Intersection` → `Daten-Server-Intersection`. Diese beschreibt, dass eine Ersetzung oder Entfernung eines Elementes aus der `Applikation-Server-Intersection` (z.B. `WebSphere App. Server`) typischerweise zu einer Anpassung der im gleichen TAP verbauten Komponenten auf der `Daten-Server-Intersection` (`z.B. IBM DB2`) führt.

Die Abbildung solcher abstrakten Anpassungsabhängigkeiten erfolgt in einem *Abhängigkeitsmodell*, welches von Domänenexperten erstellt und an ihre unternehmensindividuellen Anforderungen und Rahmenbedingungen angepasst wird. Da Experten oft über dieses spezifische Domänenwissen für Anpassungsabhängigkeiten verfügen, ist die Erstellung eines Abhängigkeitsmodells normalerweise mit wenig manuellem Aufwand verbunden. Um Experten bei der Spezifizierung von abstrakten Abhängigkeiten zu unterstützen, haben wir eine weitere DSL entwickelt. Diese ermöglicht eine formale Beschreibung des Abhängigkeitsmodells mit allen abstrakten Anpassungsabhängigkeiten, sodass diese automatisiert verarbeitet werden können. Die entwickelte DSL ist im folgenden Listing 5.3 dargestellt.

Wie im Listing 5.3 zu erkennen ist, wird ein neues Abhängigkeitsmodell mit dem Präfix `DEPENDENCY MODEL` eingeführt, gefolgt von dem String-Terminal `id` zur Definition einer beschreibenden Identifikation. Danach kann eine Reihe von abstrakten Anpassungsabhängigkeiten (`Dependency`) beschrieben werden, wobei mindestens eine zu definieren ist. Die Definition erfolgt dann über die DSL-Features `dependencyIntersection` (abhängigkeitsverursachende Intersection) und `dependentIntersection` (abhängige Intersection), welche beide mit dem Nicht-Terminal `Intersection` spezifiziert werden können. Dabei werden alle `dependentIntersection` durch die Verwendung eines Abhängigkeitspfeils ('→') einer bestimmten `dependencyIntersection` zugewiesen. Die abhängigen Intersections werden rechts von diesem Pfeil nacheinander und durch Komma separiert aufgeführt.

Eine `Intersection` wird durch die beiden Nicht-Terminale `Layer` und `Tier` definiert, welche mit einen Bindestrich ('-') verkettet werden. Diese beiden Nicht-Terminale stellen jeweils eine Auswahl an alternativen Nicht-Terminalen zur Verfügung, welche verwendet werden können, um konkrete Layer und Tiers zu beschreiben. Beispielhaft können hier der `ApplicationLayer` und der `ServerTier`

```
AbstractDependencyModel:
   'DEPENDENCY' 'MODEL' id = STRING '{'
      (dependencies += Dependency)+
   '}'
;

Dependency:
   dependancyIntersection = Intersection '->' '{'
      dependentIntersections += Intersection (','
      dependentIntersections += Intersection)* '}'
;

Intersection:
   layer = Layer '-' tier = Tier
;

Layer:
   HardwareLayer | OperatingSystemLayer
      | DataStorageLayer | ApplicationLayer
      | PresentationLayer
;

Tier:
   ClientTier | ServerTier
;

ApplicationLayer:
   {ApplicationLayer} 'application'
;

ServerTier:
   {ServerTier} 'server'
;

[..]
```

Listing 5.3: Auszug aus der DSL zur Spezifizierung von abstrakten Anpassungsabhängigkeiten

genannt werden, welche mit den jeweiligen Schlüsselwörtern `application` und `server` definiert werden. Solche Aliase für die entsprechenden Layer und Tiers können in der DSL selbst oder einmalig in einer Konfigurationsdatei festgelegt werden, um für alle entworfenen DSLs gleich zu gelten.

Das nachfolgende Listing 5.4 zeigt ein auf dieser DSL basierendes Beispiel für ein Abhängigkeitsmodell mit zwei spezifizierten abstrakten Anpassungsabhängigkeiten. Dabei stellt die erste Abhängigkeit `application-server` → `data-server` die beschriebene abstrakte Anpassungsabhängigkeit unseres fortlaufenden Beispiels dar. Zudem wurde eine zweite abstrakte Anpassungsabhängig-

```
DEPENDENCY MODEL "Exemplary Dependency Model" {
   application-server -> {
      data-server, presentation-client }
}
```

Listing 5.4: Beispiel für ein spezifiziertes Abhängigkeitsmodell

keit `application-server` → `presentation-client` definiert, welche eine Anpassung von Elementen auf der `Präsentation-Client-Intersection` für die gleiche verursachende `Applikation-Server-Intersection` beschreibt.

Ausgehend von solch einem spezifizierten Abhängigkeitsmodell kann nun für jede Restrukturierungsempfehlung analysiert werden, an welchen Komponenten der betroffenen TAPs ein möglicher Anpassungsbedarf entsteht. Hierzu müssen aus den abstrakten Anpassungsabhängigkeiten des Abhängigkeitsmodells konkrete Anpassungsabhängigkeiten zwischen den Komponenten bestimmt werden, die im Rahmen einer konkreten Restrukturierungsempfehlung betroffen sind.

Dazu wird in einem ersten Schritt festgestellt, in welcher Intersection sich ein zu ersetzendes und zu entfernendes Element einer Restrukturierungsempfehlung befindet. In unserer exemplarischen Empfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` trifft dies auf die Komponente `WebSphere App. Server` zu, welche in der `Applikation-Server-Intersection` verbaut ist. Im nächsten Schritt wird dann überprüft, ob das spezifizierte Abhängigkeitsmodell für die festgestellte Intersection eine oder mehrere abstrakte Anpassungsabhängigkeiten enthält. In unserem Beispiel sind hiervon die abhängigen Intersections `Daten-Server` und `Präsentation-Client` betroffen (vgl. Listing 5.4). Im dritten Schritt wird anschließend untersucht, ob zusammen mit der betrachteten Komponente (z.B. `WebSphere App. Server`) Elemente verbaut sind, die sich in einer der identifizierten abhängigen Intersections befinden. Zu diesem Zweck wird wie bei der Identifizierung technischer Abhängigkeiten (vgl. Unterabschnitt 4.2.4) der ursprüngliche VKG der entsprechenden Komponente analysiert. Hierdurch lässt sich feststellen, welche Komponenten in den abhängigen Intersections implementiert sind und in welchen TAPs sie sich befinden. Für diese Komponenten können nun konkrete Anpassungsabhängigkeiten ermittelt werden. So können im Rahmen unseres fortlaufenden Beispiels die konkreten Anpassungsabhängigkeiten `WebSphere App. Server` → `IBM DB2` und `WebSphere App. Server` → `Firefox` identifiziert werden. Dies kann beispielsweise andeuten, dass Konfigurationsdateien für die Datenbank angepasst und weitere Plugins oder Erweiterungen zum Browser hinzugefügt werden müssen.

Bei solch einer konkreten Anpassungsabhängigkeit kann es sich um eine *bekannte* oder *unbekannte* Abhängigkeit handeln, was Auswirkungen auf die nachfolgende Aufwandsschätzung (vgl. Unterabschnitt 5.1.3) hat. Um dies unterscheiden zu können, wird in einem letzten Schritt die *Verwendung* der abhängigen Komponente anhand

ihres entsprechenden VKGs analysiert. Diese Analyse erfolgt genau wie bei der vorgestellten Identifizierung von technischen Abhängigkeiten (vgl. Unterabschnitt 4.2.4).

Definition 5.3: Bekannte und unbekannte Anpassungsabhängigkeit

Eine *konkreten Anpassungsabhängigkeit* (Komponente B → Komponente C) wird als *bekannt* bezeichnet, wenn die abhängige Komponenten C bereits in einem anderen TAP zusammen mit dem ersetzenden Element A (`A ERSETZT B`) oder ohne das zu entfernenden Element B (`ENTFERNUNG B`) der betroffenen Restrukturierungsempfehlung `REi` verbaut wurde. Ist dies nicht der Fall, handelt es sich um eine *unbekannte* Anpassungsabhängigkeit.

Demnach liegt beispielsweise eine bekannte Anpassungsabhängigkeit vor, wenn `WebSphere App. Server` → `IBM DB2` eine konkrete Anpassungsabhängigkeit für die Restrukturierungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` darstellt und die Komponente `IBM DB2` in einem anderem TAP, in dem nicht der `WebSphere App. Server` eingesetzt wird, zusammen mit der Komponente `Tomcat` implementiert ist. Dies bedeutet, dass das notwendige technische Wissen im Unternehmen vorhanden ist, die abhängige Komponente `IBM DB2` so anzupassen, dass sie mit der zukünftig zu nutzenden Komponente `Tomcat` reibungslos funktioniert. Ähnlich verhält es sich bei einer zu entfernenden Komponente. Existiert beispielsweise die konkrete Anpassungsabhängigkeit `phpMyAdmin` → `MySQL` für eine Entfernungsempfehlung `RE2 (ENTFERNUNG phpMyAdmin)`, so wird überprüft, ob es einen anderen TAP gibt, der die Komponente `MySQL` ohne die Komponente `phpMyAdmin` beinhaltet. Ist dies der Fall, so wird auch diese konkrete Anpassungsabhängigkeit als bekannt betrachtet, da das technische Wissen zur Verwendung der abhängigen Komponente `MySQL` ohne die zu entfernende Komponente `phpMyAdmin` vorliegt. Im umgekehrten Fall würden die beschriebenen Anpassungsabhängigkeiten als unbekannt betrachtet werden. Dies beschreibt die Tatsache, dass es bisher keinen analysierten TAP gibt, in dem die betrachtete Anpassung bereits erfolgreich umgesetzt worden ist. Insofern wird davon ausgegangen, dass das technische Wissen für die Umsetzung der beabsichtigten Anpassung möglicherweise nicht vorhanden ist.

Im Rahmen dieser Anpassungsbedarfsanalyse wird ebenfalls das Ziel verfolgt, einen Ansatz auf Basis der verfügbaren TAPs-Daten zu entwickeln, der den manuellen Aufwand so gering wie möglich hält. Da die Daten der zugrunde liegenden TAPs allerdings keine Informationen über konkrete Anpassungsabhängigkeiten beinhalten, können, ähnlich wie bei den technischen Abhängigkeiten (vgl. Unterabschnitt 4.2.4), lediglich Potentiale für Anpassungsabhängigkeiten identifiziert werden, welche anschließend durch Domänenexperten verifiziert werden müssen. Dazu werden die identifizierten bekannten und unbekannten Anpassungsabhängigkeiten als Ergebnis dieser Analyse an die entsprechende Restrukturierungsempfehlung angehangen, sodass Experten diese im Rahmen der Entscheidungsfindung (vgl. Abschnitt 5.3) manuell

überprüfen können. Da diese Überprüfung lediglich für die Restrukturierungsempfehlungen erfolgen muss, welche Experten für eine Umsetzung in Erwägung ziehen, lässt sich so der manuelle Aufwand zur Verifizierung gering halten.

5.1.3 Aufwandsschätzung

Damit Domänenexperten angemessene Entscheidungen zur Restrukturierung treffen können, ist es ebenfalls erforderlich, den potentiellen Aufwand zu betrachten, der mit der Umsetzung einer konkreten Restrukturierungsempfehlung einhergeht. In diesem Kontext unterscheiden wir zwei verschiedene Aufwandsarten: den *Migrationsaufwand* und den *Anpassungsaufwand*.

Definition 5.4: Migrationsaufwand

Der *Migrationsaufwand* beschreibt den Arbeitsaufwand für alle Restrukturierungstätigkeiten, welche direkt für die Umsetzung einer spezifischen Restrukturierungsempfehlung erforderlich sind.

So umfasst der Migrationsaufwand alle Aktivitäten zur Restrukturierung von betrachteten TAPs im Hinblick auf eine konkrete Ersetzungs- oder Entfernungsempfehlung. Dazu zählen beispielsweise die Deinstallation einer zu entfernenden Komponente und die Löschung der mit ihr verbundenen Daten sowie die Installation und Konfiguration einer neu zu verwendenden Komponente. Somit entsteht der Migrationsaufwand für die betroffenen Komponenten einer spezifischen Restrukturierungsempfehlung `REi` (z.B. `Tomcat` und `WebSphere App. Server` für `RE1`).

Definition 5.5: Anpassungsaufwand

Der *Anpassungsaufwand* beschreibt den Arbeitsaufwand für alle Anpassungstätigkeiten, welche sich indirekt aus der Umsetzung einer spezifischen Restrukturierungsempfehlung ergeben und für die reibungslose Funktion des restrukturierten Systems erforderlich sind.

Solche Anpassungstätigkeiten können beispielsweise das notwendige Update einer Komponente zur neusten Version oder die erforderliche Anpassung von Schnittstellen und Konfigurationen von bereits vorhandenen Komponenten darstellen. Dabei entsteht der Anpassungsaufwand einer spezifische Restrukturierungsempfehlung `REi` bei solchen Elementen eines TAPs, für die eine konkrete Anpassungsabhängigkeit in Bezug auf `REi` identifiziert worden ist (z.B. `IBM DB2` für `RE1`).

Zur automatischen Abschätzung der Migrations- und Anpassungsaufwände, die mit einer spezifischen Restrukturierungsempfehlung verbunden sind, wird eine von Experten zu spezifizierende *Aufwandsmatrix* verwendet. Wie in Tabelle 5.1

	Client	Server
Präsentation	2	2
Applikation	3	5
Daten	2	4
Betriebssystem	2	2
Hardware	1	2

1 = sehr niedrig *2* = niedrig *3* = mittel *4* = hoch *5* = sehr hoch

Tabelle 5.1: Aufwandsmatrix mit beispielhaften Aufwandsschätzungen

dargestellt ist, weist solch eine Aufwandsmatrix die gleiche Struktur (mit allen betrachteten Layern und Tiers) wie die analysierten TAPs auf. Für jede einzelne Intersection beinhaltet diese Matrix einen durchschnittlichen *Aufwandswert AW* zwischen 1 und 5, welche von Domänenexperten zu schätzen sind. Dabei beschreibt solch ein Wert eine durchschnittlich geschätzte *Aufwandsklasse*.

Eine Aufwandsklasse dient als Basis für die Bestimmung von möglichen Migrations- und Anpassungsaufwänden, die bei der Umsetzung einer Restrukturierungsempfehlung innerhalb einer konkreten Intersection zu erwarten sind. Dabei beschreibt solch eine Klasse den Aufwand in Form von Kategorien. So stellt der Wert 1 die Kategorie *sehr niedrig*, der Wert 3 die Kategorie *mittel* und der Wert 5 die Kategorie *sehr hoch* dar. Diese Werte sind für jede Intersection durch Domänenexperten zu schätzen und in einer Aufwandsmatrix zu spezifizieren. In diesem Zusammenhang sollten mehrere Experten unabhängig voneinander eine Aufwandsschätzung unter Berücksichtigung der unternehmensindividuellen Bedingungen durchführen. Durch eine anschließende Mittelwertbildung können aussagekräftige Aufwandsindikatoren abgeleitet werden.

Dabei liegt der Hauptzweck solch einer Aufwandsmatrix nicht darin, konkrete Aufwände automatisiert zu identifizieren. Dies wäre so nicht möglich, da hierfür nicht alle notwendigen Daten (z.B. Informationen über alle erforderlichen Tätigkeiten und deren konkreter Arbeitsaufwand) in den analysierten TAPs zur Verfügung stehen. Vielmehr soll die automatisierte Aufwandsschätzung Experten dabei helfen, die potentiellen Aufwände verschiedener Restrukturierungsempfehlungen miteinander vergleichen zu können, um so geeignete Restrukturierungen zu identifizieren, die mit den verfügbaren Mitteln in einem Unternehmen auch umsetzbar sind.

Um auf Basis einer Aufwandsmatrix durchschnittliche Migrations- und Anpassungsaufwände für jede Restrukturierungsempfehlung automatisiert bestimmen zu können, muss diese in einer formalen Beschreibung vorliegen. Zu diesem Zweck haben wir eine weitere DSL entworfen, die im nachfolgenden Listing 5.5 gezeigt wird. Wie hier zu erkennen ist, wird eine neue Aufwandsmatrix mit Hilfe des Präfix `EFFORT MATRIX`, gefolgt von einem Bezeichner (`id`), eingeführt. Anschließend

```
EffortMatrix:
   'EFFORT' 'MATRIX' id = STRING '{'
      (efforts += Effort)+
   '}'
;

Effort:
   layer = Layer '{'
      tierValues += TierValueElement
         (',' tierValues += TierValueElement)*
   '}'
;

TierValueElement:
   tier = Tier ':' value = INT
;

[..]
```

Listing 5.5: Auszug aus der DSL zur Spezifizierung einer Aufwandsmatrix

kann der Aufwand (`Effort`) für mindestens einen `Layer` definiert werden, indem diesem Layer ein oder mehrere Elemente vom Typ `TierValueElement` zugeordnet werden. Dieses Element stellt ein Nicht-Terminal dar, welches wiederum durch das Nicht-Terminal `Tier` und dem `INT`-Terminal `value` definiert ist. So lässt sich eine Aufwandsklasse für einen betroffenen `Layer` durch Spezifizierung des entsprechenden `Tier` und Zuweisung (`:`) eines nummerischen Wertes (`value`) beschreiben.

Im folgenden Listing 5.6 ist eine exemplarische Aufwandsmatrix dargestellt. Diese beinhaltet die Spezifikation von durchschnittlichen Aufwandswerten für die unterschiedlichen Intersections unseres fortlaufenden Beispiels. Darin sind beispielsweise folgende Aufwandswerte definiert: `Applikation-Client=3`, `Applikation-Server=5`, `Daten-Client=2` und `Daten-Server=4`. Daraus lässt sich unter anderem ableiten, dass der durchschnittliche Aufwand innerhalb der Intersection `Applikation-Server` sehr hoch ist und mit einem Wert von 5 höher liegt, als der Aufwand in allen anderen Intersections. Folglich ist die Umsetzung von Restrukturierungsempfehlungen innerhalb dieser Intersection mit dem höchsten Aufwand verbunden.

```
EFFORT MATRIX "Exemplary Effort Matrix" {
   presentation {client: 2, server: 2}
   application {client: 3, server: 5}
   data {client: 2, server: 4}
   operatingSystem {client: 2, server: 2}
   hardware {client: 1, server: 2}
}
```

Listing 5.6: Beispielhafte Aufwandsmatrix

Um nun auf Basis solch einer Aufwandsmatrix den *Migrationsaufwand* A_M einer spezifischen Restrukturierungsempfehlung `REi` zu bestimmen, wird die folgende Gleichung 5.2 angewendet.

$$A_M(REi) = AW(I) * N_K * N_{T(k_n)} \tag{5.2}$$

Hiernach wird der *Migrationsaufwand* $A_M(REi)$ ermittelt, indem der Aufwandswert AW der betroffenen Intersection I mit der Anzahl N der zu restrukturierenden physischen Komponenten K und der Anzahl aller betroffenen TAPs T, welche eine Komponente $k_n \in K$ beinhalten, multipliziert wird. So können Restrukturierungsempfehlungen berücksichtigt werden, die entweder ein einzelnes Element (z.B. bei Entfernungsempfehlungen) oder mehrere solcher Elemente (z.B. bei Ersetzungsempfehlungen für mehrere Komponenten) betrachten. Außerdem fließen auch unterschiedliche Instanzen der gleichen Komponente k_n in verschiedenen TAPs $T(k_n)$ mit in die Berechnung ein, da eine Umstrukturierung für alle Instanzen eines zu restrukturierenden Elementes durchgeführt werden muss.

Für die Empfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` aus unserem fortlaufenden Beispiel ergibt sich somit der geschätzte Migrationsaufwand A_M(RE1)=5∗1∗1=5, da für die `Applikation-Server-Intersection` der Aufwandswert AW=5 gilt (vgl. Listing 5.6) und durch `RE1` lediglich das einzelne Element `WebSphere App. Server` im `TAP3` (vgl. Tabelle 3.1) betroffen ist.

Die Bestimmung des *Anpassungsaufwandes* A_A einer spezifische Restrukturierungsempfehlung `REi` erfolgt ebenfalls auf Basis einer definierten Aufwandsmatrix nach der folgenden Gleichung 5.3:

$$A_A(REi) = \sum_{j=1}^{n} (AW(I(ak_j)) * f(ak_j) * N_{T(ak_j)}) \tag{5.3}$$

Dabei beschreibt $ak_j \in AK$ eine abhängige Komponente ak aus der Menge AK aller Komponenten, die eine Anpassungsabhängigkeit mit dem zu entfernenden Kandidaten einer konkreten Restrukturierungsempfehlung `REi` aufweisen. Darüber hinaus definiert f einen Faktor, der zur Unterscheidung von bekannten und unbekannten Anpassungsabhängigkeiten dient und frei konfigurierbar ist. Im Kontext unserer Arbeit haben sich die Werte $f = 0,5$ für bekannte Anpassungsabhängigkeiten und $f = 1$ für unbekannte Anpassungsabhängigkeiten bewährt (vgl. Design-Entscheidung 5.1).

So errechnet sich der Anpassungsaufwand A_A für `REi` durch die Summe der Einzelanpassungsaufwände für alle $ak_j \in AK$. Solch ein Einzelaufwand wird demnach durch die Multiplikation des Aufwandswertes AW der von ak_j betroffenen Intersection I mit dem zu ak_j passendem Faktor f und der Anzahl $N_{T(ak_j)}$ von TAPs, die ak_j implementieren, ermittelt.

Design-Entscheidung 5.1: Definierte Werte für den Faktor *f*

Der Faktor *f* ermöglicht eine unterschiedliche Aufwandsschätzung für bekannte und unbekannte anpassungsabhängige Elemente. In Gesprächen mit Experten auf Basis von Industriedaten konnten die Werte *f=0,5* für bekannte und *f=1* für unbekannte Anpassungsabhängigkeiten ermittelt werden. Dies drückt aus, dass der Aufwand für unbekannte Anpassungsmaßnahmen durchschnittlich doppelt so hoch ist wie für bekannte, da das technische Wissen hierfür oft nicht sofort verfügbar ist. Somit muss dieses erst durch z.B. externe Beratung, Schulung oder eigene Entwicklungstätigkeit aufgebaut werden. Allerdings kann der Faktor *f* an unternehmensindividuelle Bedingungen angepasst werden.

Für `RE1 (Tomcat ERSETZT WebSphere App. Server)` als exemplarische Restrukturierungsempfehlung unseres fortlaufenden Beispiels wurden die beiden anpassungsabhängigen Komponenten ak_1 = `IBM DB2` und ak_2 = `Firefox` identifiziert. In beiden Fällen handelt es sich um eine bekannte Anpassungsabhängigkeit. Daraus ergibt sich der Anpassungsaufwand $A_A(RE1)= 4*0{,}5*1 + 2*0{,}5*1 =3$, mit $AW(I(ak_1))=4$, $AW(I(ak_2))=2$ und $N_{T(ak_1)}=N_{T(ak_2)}=1$ (`TAP3`).

Nach erfolgter Aufwandsschätzung werden die ermittelten Ergebnisse für den Migrationsaufwand A_M und den Anpassungsaufwand A_A an die entsprechende Restrukturierungsempfehlung `REi` angehangen, damit auch diese für nachfolgenden Analysen und die spätere Entscheidungsfindung zur Verfügung stehen.

5.2 Bewertung des Portfolios von Restrukturierungsempfehlungen

Nachdem alle Restrukturierungsempfehlungen einzeln bewertet worden sind, können diese miteinander verglichen werden, wodurch eine Bewertung des Gesamtportfolios möglich ist. Diese Bewertung erfolgt in zwei unterschiedlichen Analyseschritten, wie in der folgenden Abbildung 5.4 zu erkennen ist.

Zuerst wird eine *Korrelationsanalyse* (vgl. Unterabschnitt 5.2.1) durchgeführt, die unterschiedliche Typen von Korrelationen zwischen einzelnen Restrukturierungsempfehlungen identifiziert. So lässt sich feststellen, welche Restrukturierungsempfehlungen zusammen umgesetzt werden können und welche nicht. Im Rahmen der *Portfolioanalyse* (vgl. Unterabschnitt 5.2.2) erfolgt anschließend die Erstellung einer Heatmap und einer Aufwand-Nutzen-Matrix, welche bei der Bestimmung geeigneter Restrukturierungsempfehlungen unterstützen.

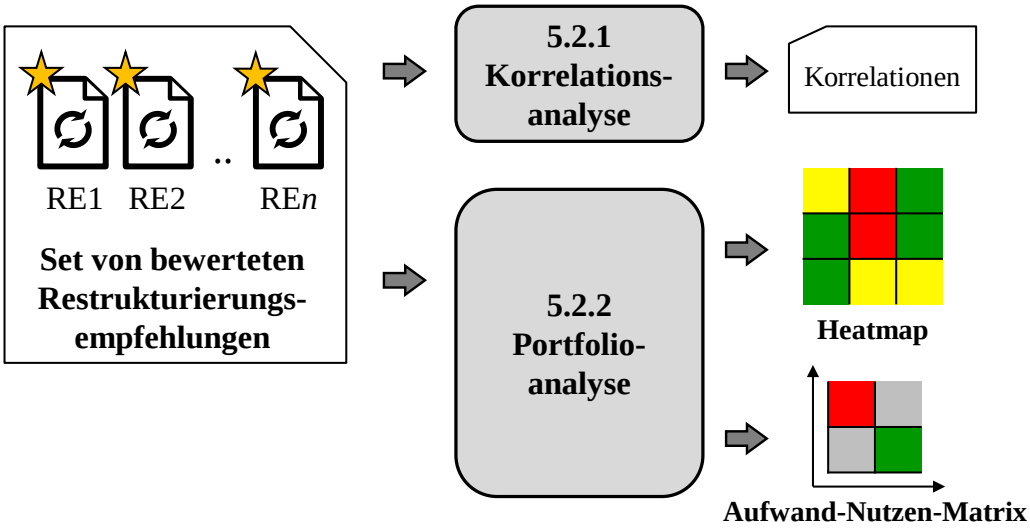


Abbildung 5.4: Workflow für die Bewertung des Gesamtportfolios

5.2.1 Korrelationsanalyse

Bei der übergreifenden Betrachtung über alle Intersections der analysierten TAPs hinweg lassen sich Korrelationen zwischen den abgeleiteten Restrukturierungsempfehlungen erkennen, die Aufschluss darüber geben, welche von ihnen zusammen umgesetzt werden können und welche nicht. Wie in Abbildung 5.5 dargestellt ist, gibt es drei Arten von Korrelationen. Unterschiedliche Ersetzungsempfehlungen können *verwandt*, *widersprüchlich* oder *transitiv* zueinander sein. Dagegen können zwischen einzelnen Ersetzungsempfehlungen nur *verwandte* Korrelationen bestehen und zwischen Ersetzungs- und Entfernungsempfehlungen entweder *verwandte* oder *widersprüchliche* Korrelationen existieren.

Wie die nachfolgende Abbildung 5.5 zeigt, sind zwei Ersetzungsempfehlungen dann verwandt zueinander, wenn beide das gleiche zu implementierende Element `A` berücksichtigen. Dies bedeutet, dass zwei verschiedene Komponenten `B` und `C` durch dasselbe Element `A` ersetzt werden. Da dies unabhängig voneinander möglich ist, können beide Ersetzungsempfehlungen gleichzeitig umgesetzt werden. So wären beispielsweise die zwei Ersetzungsempfehlungen `RE1 (Tomcat ERSETZT WebSphere App. Server)` und `RE2 (Tomcat ERSETZT Jetty)` verwandt, da sie beide beabsichtigen, die Komponente `Tomcat` zu implementieren. Ähnlich verhält es sich bei verwandten Entfernungsempfehlungen sowie bei Ersetzungs- und Entfernungsempfehlungen mit diesem Korrelationstyp. Solche Restrukturierungsempfehlungen sind miteinander verwandt, wenn sie beabsichtigen, das gleiche Element `B` zu eliminieren. Demnach sind beispielsweise `RE1 (Tomcat ERSETZT WebSphere App. Server)` und `RE6 (ENTFERNUNG WebSphere App. Server)` verwandt und können zusammen realisiert werden. Ist hierdurch jedoch die gleiche Intersection betroffen, würde `RE1` nicht umgesetzt werden, da das zu ersetzende Element durch `RE6` entfernt wird.

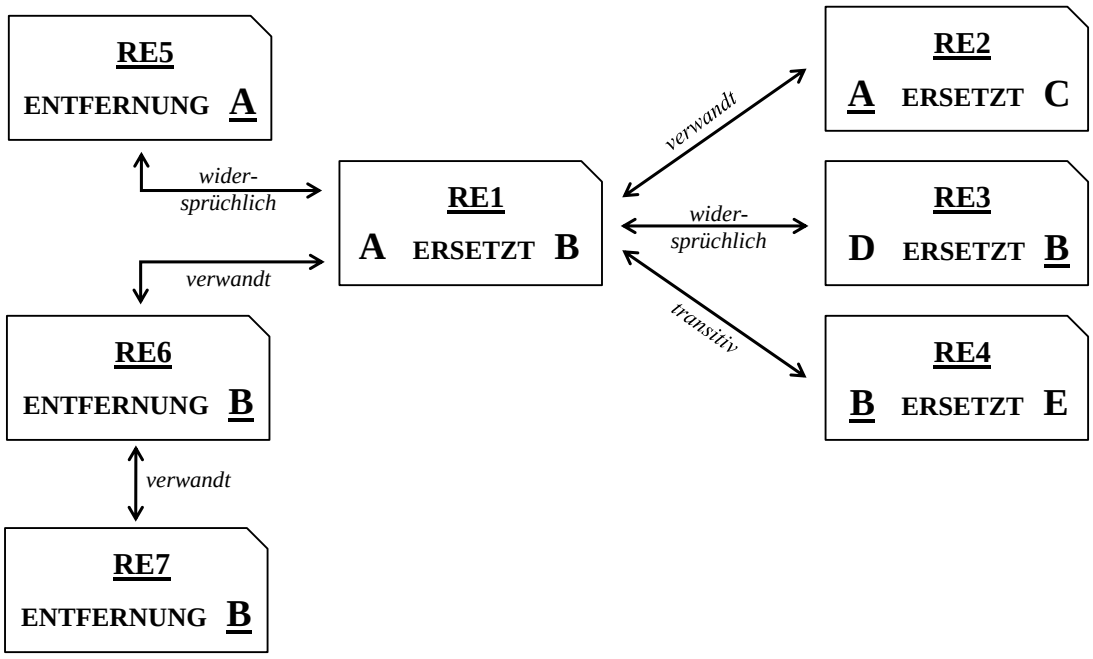


Abbildung 5.5: Korrelationen zwischen Restrukturierungsempfehlungen

Im Gegensatz dazu sind zwei Ersetzungsempfehlungen widersprüchlich zu einander, wenn beide dasselbe zu restrukturierende Element `B` fokussieren. Dies würde dazu führen, dass das gleiche Element `B` durch zwei unterschiedliche Elemente `A` und `D` ersetzt wird. Wenn beide Ersetzungsempfehlungen dabei auf die gleiche Intersection abzielen, können nicht beide umgesetzt werden, da das Element `B` innerhalb einer Intersection nur durch eines dieser beiden Elemente ausgetauscht werden kann. Zielen sie dagegen auf unterschiedliche Intersections, wären sie technisch zwar beide umsetzbar, würden aber zu widersprüchlichen Architekturentscheidungen führen. Beispielsweise wäre eine dritte Ersetzungsempfehlung `RE3 (Glassfish ERSETZT WebSphere App. Server)` widersprüchlich zu `RE1 (Tomcat ERSETZT WebSphere App. Server)`, da die Komponente `WebSphere App. Server` entweder durch `Tomcat` oder durch `Glassfish` ersetzt werden sollte. Existiert darüber hinaus eine Entfernungsempfehlung (z.B. `RE5 (ENTFERNUNG Tomcat)`), welche beabsichtigt, das zu implementierende Element `A` einer Ersetzungsempfehlung (z.B. `RE1 (Tomcat ERSETZT WebSphere App. Server)`) zu eliminieren, so liegt auch zwischen diesen Empfehlungen eine widersprüchliche Korrelation vor.

Darüber hinaus kann es auch transitiv verbundene Ersetzungsempfehlungen geben. Dies ist der Fall, wenn eine Ersetzungsempfehlung ein zu implementierendes Element `B` beinhaltet und eine weitere Ersetzungsempfehlung beabsichtigt, das gleiche Element `B` auszutauschen. Zum Beispiel wäre `RE1 (Tomcat ERSETZT WebSphere App. Server)` transitiv zu einer weiteren

Ersetzungsempfehlung `RE4 (WebSphere App. Server ERSETZT JBoss)` verbunden, da die Komponente `WebSphere App. Server` durch `RE1` ersetzt und durch `RE4` implementiert wird. Solche transitiv verbundene Ersetzungsempfehlungen können zusammen umgesetzt werden, indem die transitive Abhängigkeit aufgelöst wird. So lassen sich aus zwei transitiven Ersetzungsempfehlungen `RE1 (A ERSETZT B)` und `RE4 (B ERSETZT E)` die verwandten Restrukturierungsempfehlungen `RE1 (A ERSETZT B)` und `RE4 (A ERSETZT E)` ableiten. Für das beschriebene Beispiel würde dies zu `RE1 (Tomcat ERSETZT WebSphere App. Server)` und `RE4 (Tomcat ERSETZT JBoss)` führen.

Die Identifizierung solcher Beziehungen erfolgt im Rahmen der automatisierten *Korrelationsanalyse*. Hierbei wird jede Restrukturierungsempfehlung daraufhin untersucht, ob diese eine oder mehrere der beschriebenen Korrelationstypen zu anderen Restrukturierungsempfehlungen aufweist. Zu diesem Zweck wird die zu implementierende und die zu entfernende Komponenten der jeweiligen Restrukturierungsempfehlung mit den Elementen aller anderen Restrukturierungsempfehlungen verglichen, um konkrete Korrelation zu identifizieren. Die so ermittelten Korrelationen werden der entsprechenden Restrukturierungsempfehlung hinzugefügt, um sie für die Entscheidungsfindung von Domänenexperten verfügbar zu machen.

5.2.2 Portfolioanalyse

Um Domänenexperten bei der Auswahl geeigneter Restrukturierungsempfehlungen zu unterstützen, werden diese im Vergleich zueinander bewertet. Dies erfolgt im Rahmen der *Portfolioanalyse*, welche das gesamte Set aller abgeleiteten Restrukturierungsempfehlungen berücksichtigt. Hierfür wird zum einen eine *Heatmap* erstellt, welche die Identifizierung von Intersections mit hohem Restrukturierungspotential ermöglicht und zum anderen eine *Aufwand-Nutzen-Matrix* generiert, welche Experten dabei hilft, Restrukturierungspotentiale mit positiven Aufwand-Nutzen-Verhältnis zu ermitteln.

Heatmap

Eine Heatmap wird durch die gleiche Struktur, wie die analysierten TAPs dargestellt. So finden sich darin alle betrachteten Layer, Tiers und Intersections wieder, wie die beispielhafte Heatmap in Tabelle 5.2 zeigt. Auf Basis solch einer Heatmap lässt sich für Domänenexperten leicht erkennen, in welchen Intersections ein kleines, mittleres oder großes Restrukturierungspotential besteht. Dies lässt sich anhand der Anzahl an Restrukturierungsempfehlungen in den jeweiligen Intersections bestimmen.

Zur Erstellung solch einer Heatmap wird für jede einzelne Intersection I_j identifiziert, wie hoch die Anzahl $N_{RE(I_j)}$ der Restrukturierungsempfehlungen (sowohl Ersetzungs- als auch Entfernungsempfehlungen) ist, die eine Umstrukturierung in der jeweiligen Intersection I_j beabsichtigen. Die so ermittelten Werte werden dann in die entspre-

	Client	Server
Präsentation	2	1
Applikation	3	8
Daten	2	6
Betriebssystem	1	13
Hardware	1	5

grün = wenige *gelb* = mittel *rot* = viele

Tabelle 5.2: Exemplarische Heatmap

chende Intersection I_j der Heatmap eingetragen. Um auf Basis all dieser Werte anschließend drei gleich große Wertbereiche (*wenige, mittel, viele*) für die Heatmap zu bestimmen, wird deren Minimalwert $a = \min(\{N_{RE(I_1)}, N_{RE(I_2)}, ...N_{RE(I_n)}\})$ sowie deren Maximalwert $b = \max(\{N_{RE(I_1)}, N_{RE(I_2)}, ...N_{RE(I_n)}\})$ ermittelt. Aus der Differenz d=b-a lassen sich dann die drei Wertbereiche mit Hilfe der drei Intervalle $I_A = [a; a + \frac{1}{3}d[$, $I_B = [a + \frac{1}{3}d; a + \frac{2}{3}d]$ und $I_C =]a + \frac{2}{3}d; b]$ ableiten. Dabei stellt I_A einen Bereich mit *wenigen* Restrukturierungsempfehlungen dar und ist grün gekennzeichnet. I_B beschreibt einen Bereich mit *mittel* und I_C einen mit *vielen* Restrukturierungsempfehlungen. Diese sind jeweils *gelb* beziehungsweise *rot* eingefärbt.

In der exemplarischen Heatmap in Tabelle 5.2 ist d=b-a=12, mit a=1 und b=13. Daraus ergeben sich die drei Bereiche $I_A = [1; 5[$, $I_B = [5; 9]$ und $I_C =]9; 13]$. Demnach bietet die rot gekennzeichnete `Betriebssystem-Server-Intersection` viele Restrukturierungsempfehlungen und ist mit einem hohen Restrukturierungspotential verbunden. Dagegen ist das Restrukturierungspotential bei den gelb gekennzeichneten Intersections (z.B. `Daten-Server`) eher mittelmäßig und in den grün gefärbten Bereichen (z.B. `Hardware-Client`) lediglich gering.

Aufwand-Nutzen-Matrix

Für die Auswahl geeigneter Restrukturierungsempfehlungen ist es erforderlich, dass Domänenexperten den Aufwand zur Umsetzung einer Restrukturierungsempfehlung im Verhältnis zu ihrem Nutzen bewerten. In diesem Zusammenhang wird die identifizierte Variabilitätsdifferenz einer Restrukturierungsempfehlung als ihr Nutzen betrachtet, da diese die erreichbare Variabilitätsreduzierung beschreibt. Für die Restrukturierung von TAPs sind vor allem solche Empfehlungen interessant, die einen hohen Nutzen haben und gleichzeitig mit wenig Aufwand umsetzbar sind. Dies wird als *positives Aufwand-Nutzen-Verhältnis* bezeichnet. Den umgekehrten Fall betrachten wir hingegen als *negatives Aufwand-Nutzen-Verhältnis*. Um Experten bei der Identifizierung von Restrukturierungsempfehlungen mit solch einem positiven Aufwand-Nutzen-Verhältnis zu unterstützen, wird eine *Aufwand-Nutzen-Matrix* erstellt, wie sie in Abbildung 5.6 zu sehen ist.

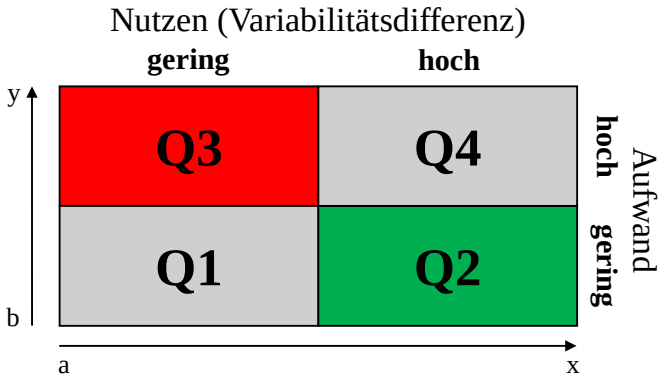


Abbildung 5.6: Darstellung einer Aufwand-Nutzung-Matrix

Solch eine Aufwand-Nutzen-Matrix berücksichtigt alle generierten Restrukturierungsempfehlungen und ordnet sie jeweils in einen ihrer vier *Quadranten Q1–Q4* ein, abhängig davon, wie *hoch* oder *gering* der mit einer spezifischen Restrukturierungsempfehlung verbundene Aufwand und Nutzen ist. So werden im *grünen Quadranten Q2* alle Restrukturierungsempfehlungen mit einem positiven Aufwand-Nutzen-Verhältnis eingeordnet. Im *roten Quadranten Q3* werden dagegen alle Restrukturierungsempfehlungen mit negativen Aufwand-Nutzen-Verhältnis einsortiert. Die übrigen beiden, in *grau* gehaltenen Quadranten beschreiben ein neutrales Verhältnis, bei dem die Werte für Aufwand und Nutzen entweder beide gering (*Q1*) oder beide hoch (*Q4*) sind.

Zur Erstellung solch einer Aufwand-Nutzen-Matrix werden die jeweiligen Werte für die Variabilitätsdifferenz VD(REi), den Migrationsaufwand A_M(REi) und den Anpassungsaufwand A_A(REi) jeder einzelnen Restrukturierungsempfehlung `REi` herangezogen. Aus den letzten beiden Werten ergibt sich durch Addition der Gesamtaufwand $A_G(REi) = A_M(REi) + A_A(REi)$.

So lassen sich die Minimalwerte $a = \min(\{VD(RE1), VD(RE2), ...VD(REn)\})$ und $b = \min(\{A_G(RE1), A_G(RE2), ...A_G(REn)\})$ bestimmen. Zudem können auch die jeweiligen Maximalwerte $x = \max(\{VD(RE1), VD(RE2), ...VD(REn)\})$ und $y = \max(\{A_G(RE1), A_G(RE2), ...A_G(REn)\})$ ermittelt werden. Unter Berücksichtigung der Differenzen d_1=x-a und d_2=y-b können die vier gleich verteilte Intervalle I_A, I_B, I_C und I_D identifizieren werden, um darauf basierend die jeweiligen Quadranten Q1–Q4 zu bestimmen. So ergibt sich $I_A = [a; a + 0,5d_1[$ für den Bereich *geringer Nutzen* und $I_B = [b; b + 0,5d_2[$ für den Bereich *geringer Aufwand*. Darüber hinaus beschreibt $I_C = [a + 0,5d_1; x]$ den Bereich mit *hohem Nutzen* und $I_C = [b + 0,5d_2; y]$ den Bereich mit *hohem Aufwand*.

Aus der Ermittlung der Intervalle für die jeweiligen Quadranten einer Aufwand-Nutzen-Matrix wird ersichtlich, dass ein positives Aufwand-Nutzen-Verhältnis nicht für eine einzelne Restrukturierungsempfehlung bestimmt werden kann, sondern im-

mer der Vergleich mit allen anderen Restrukturierungsempfehlungen erforderlich ist. Nur so kann gewährleistet werden, dass Domänenexperten eine gute Übersicht über alle betrachteten Restrukturierungsempfehlungen erhalten und daraus leicht diejenigen auswählen können, welche insgesamt über ein positives Aufwand-Nutzen-Verhältnis verfügen.

Neben solchen Empfehlungen können aber durchaus auch Restrukturierungsempfehlungen mit neutralem Aufwand-Nutzen-Verhältnis für die Umsetzung geeignet sein, wenn sie unternehmensspezifische Architekturentscheidungen unterstützen. Dies kann beispielsweise eine strategische Entscheidung wie die *Standardisierung von SAP-Produkten* betreffen. Daher sollten solche Restrukturierungsempfehlungen von Domänenexperten im Hinblick auf ihre Unterstützung von Architekturentscheidungen bewertet werden. Darüber hinaus kann es auch vorkommen, dass Restrukturierungsempfehlungen mit negativen Aufwand-Nutzen-Verhältnis für eine Umsetzung in Frage kommen, wenn sie unbedingt notwendige Restrukturierungen unterstützen. Dies betrifft beispielsweise die *Ersetzung veralteter Technologien (z.B. Mainframe-Systeme)*, da diese oft nicht mehr vom Hersteller unterstützt und mit notwendigen Updates versorgt werden. Auch dies muss durch Domänenexperten bewertet und in die Entscheidungsfindung einbezogen werden.

5.3 Entscheidungsfindung für die Restrukturierung

Basierend auf den Ergebnissen der Einzel- und Portfoliobewertung, können nun konkrete Entscheidungen zur Restrukturierung von analysierten TAPs getroffen werden. Um Domänenexperten bei dieser Entscheidungsfindung zu unterstützen, präsentieren wir einen iterativen Entscheidungsprozess, der in Abbildung 5.7 dargestellt ist.

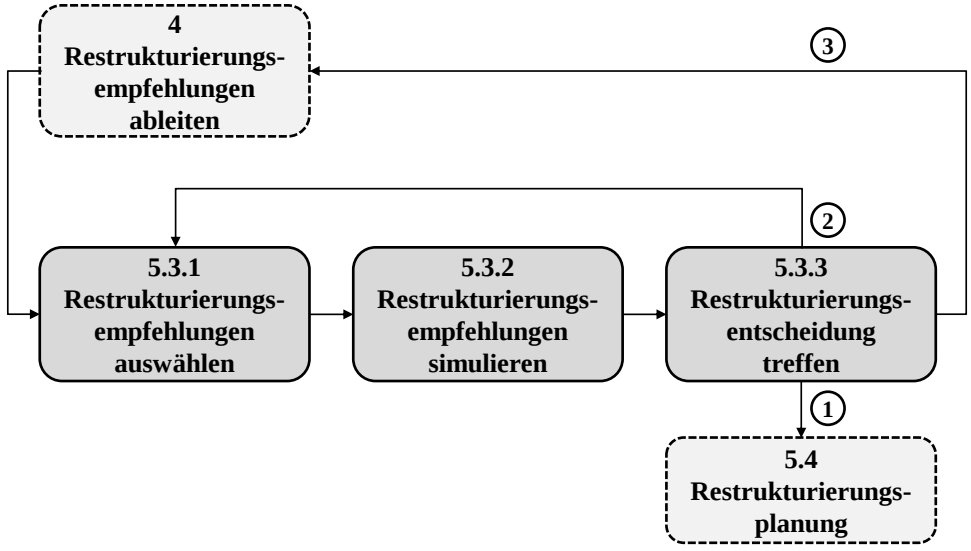


Abbildung 5.7: Iterativer Entscheidungsprozess für die Restrukturierung

Zuerst müssen Experten auf Grundlage aller zur Verfügung stehenden Informationen und ihrem eigenen Domänenwissen geeignete *Restrukturierungsempfehlungen auswählen* (vgl. Unterabschnitt 5.3.1). Diese können dann in einem nächsten Schritt *simuliert* werden (vgl. Unterabschnitt 5.3.2), indem das für alle ausgewählte Restrukturierungsempfehlungen resultierende Soll-150%-Modell erzeugt wird. Auf Basis dieses Soll-150%-Modell kann dann über die konkreten Restrukturierungen entschieden werden (vgl. Unterabschnitt 5.3.3). Hierbei gibt es drei Entscheidungsalternativen: erstens, die Umsetzung der ausgewählten Empfehlungen und die anschließende Restrukturierungsplanung (vgl. Abschnitt 5.4); zweitens, die Auswahl anderer Restrukturierungsempfehlungen; und drittens, die Ableitung neuer Restrukturierungsempfehlungen für das generierte Soll-150%-Modell (vgl. Kapitel 4).

5.3.1 Auswahl geeigneter Restrukturierungsempfehlungen

Um eine Reduzierung der Variabilität von analysierten TAPs zu erreichen, ist zuerst eine Entscheidung über deren Restrukturierung erforderlich. Zu diesem Zweck können Domänenexperten eine oder mehrere Restrukturierungsempfehlungen, welche nach ihrer Einschätzung für eine Restrukturierung geeignet sind, aus dem Set aller bewerteten Restrukturierungsempfehlungen auswählen.

In diesem Kontext wird eine Restrukturierungsempfehlung dann als *geeignet* betrachtet, wenn sie umsetzbar ist und dabei keinem allgemeinen Architekturziel[2] des betrachteten Unternehmens widerspricht. Dabei wird unter *umsetzbar* verstanden, dass die beabsichtigte Restrukturierung erfolgreich unter Berücksichtigung ihrer Abhängigkeiten und Korrelationen durchgeführt werden kann.

Zur Identifizierung und Auswahl von geeigneten Restrukturierungsempfehlungen stehen Domänenexperten die im Rahmen unseres Ansatzes erzeugten Informationen zur Verfügung. Einen Überblick darüber gibt die folgende Abbildung 5.8. Wie hier zu erkennen ist, können Domänenexperten für die Auswahl von geeigneten Restrukturierungsempfehlungen auf die jeweiligen Ergebnisse aus der Einzel- und Portfoliobewertung sowie auf das generierte 150%-Modell mit allen seinen Variabilitätsinformationen zugreifen. Dabei ist die folgende Vorgehensweise zu empfehlen:

1. Restrukturierungsbereich festlegen: Zuerst sollte ein Bereich festgelegt werden, der für eine Restrukturierung in Frage kommt. Dabei steht die Auswahl von ein oder mehreren Intersections der entsprechenden TAPs im Fokus. Hierfür können Domänenexperten das generierte *150%-Modell* verwenden, um sich einen ersten Überblick über alle Intersections und ihre jeweilige Variabilität zu schaffen. Anschließend lässt sich mit Hilfe der erzeugten *Heatmap* feststellen, welche der betrachteten Intersections das größte Restrukturierungspotential bieten. So können Domänenexperten auf

[2] Allgemeine IT-Architekturziele sind beispielsweise die Minimierung der Kosten, die Maximierung der Flexibilität sowie die Maximierung der Qualität [Kel12].

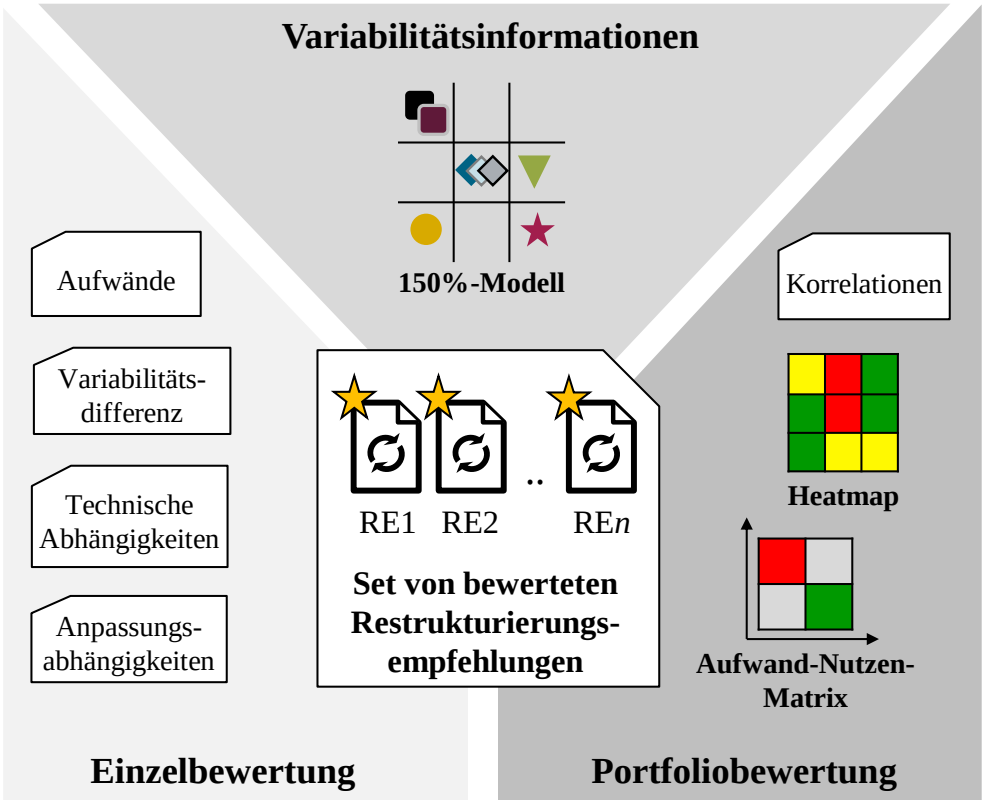


Abbildung 5.8: Verfügbare Informationen für die Auswahl von geeigneten Restrukturierungsempfehlungen

dieser Basis entscheiden, welche Intersections als Restrukturierungsbereich festgelegt und näher untersucht werden sollen.

2. Nutzbringende Restrukturierungsempfehlungen bestimmen: Nach Festlegung des Restrukturierungsbereiches können jene Restrukturierungsempfehlungen im Gesamtportfolio bestimmt werden, die eine Restrukturierung in den vorgesehenen Intersections beabsichtigen. Durch Hinzunahme der *Aufwand-Nutzen-Matrix* kann festgestellt werden, welche dieser Restrukturierungsempfehlungen ein positives Aufwand-Nutzen-Verhältnis haben und damit interessant für eine Restrukturierung sind. Für diese kann weiterhin ein detaillierter Vergleich zwischen ihrem Aufwand und Nutzen durchgeführt werden, indem ihre einzelnen Werte für *Migrations-* und *Anpassungsaufwand* sowie *Variabilitätsdifferenz* betrachtet werden. Zudem können auch Empfehlungen mit neutralem oder negativen Aufwand-Nutzen-Verhältnis ausgewählt werden, wenn ihre Restrukturierung nutzbringend in Bezug auf ein vorliegendes Architekturziel ist (z.B. der Austausch von veralteten Technologien).

3. Geeignete Restrukturierungsempfehlungen auswählen: Zuletzt müssen die nutzbringenden Restrukturierungsempfehlungen noch in Bezug auf ihre Abhängigkei-

ten und Korrelationen betrachtet werden. Dazu sollten Domänenexperten zuerst prüfen, ob zwischen den angedachten Restrukturierungsempfehlungen widersprüchliche *Korrelationen* bestehen. Ist dies der Fall, muss eine der verursachenden Restrukturierungsempfehlungen aussortiert werden, um den angezeigten Widerspruch aufzulösen. Anschließend können die Abhängigkeiten für alle übrigen Restrukturierungsempfehlungen untersucht werden. Durch die Verifizierung von *technischen Abhängigkeiten* können Domänenexperten konkrete Inkompatibilitäten erkennen und betroffene Restrukturierungsempfehlungen ebenfalls aussortieren. Zudem kann der jeweilige Anpassungsbedarf durch die Verifizierung der einzelnen *Anpassungsabhängigkeiten* genauer bestimmt werden. Hierdurch kann sich der entsprechende Anpassungsaufwand ändern, was nochmals zur Überprüfung des Aufwand-Nutzen-Verhältnis führen sollte. Zum Schluss können alle nicht aussortierten Restrukturierungsempfehlungen als geeignet betrachtet werden, sodass Domänenexperten diese oder eine Auswahl hieraus für die Restrukturierung selektieren können.

5.3.2 Simulation der ausgewählten Restrukturierungsempfehlungen

Nachdem Domänenexperten geeignete Restrukturierungsempfehlungen bestimmt haben, können diese auf Basis des Ist-150%-Modells simuliert werden. Dies erfolgt in unserem Ansatz durch die automatische Transformation des 150%-Modells auf Basis einer oder mehrerer ausgewählter Restrukturierungsempfehlungen. Hierdurch entsteht als Ergebnis ein Soll-150%-Modell, welches die resultierende Soll-Architektur beschreibt und für Domänenexperten visuell präsentiert wird. Durch dieses Soll-150%-Modell wird ersichtlich, wie sich die Variabilität in den betrachteten TAPs tatsächlich aufgrund der ausgewählten Restrukturierungsempfehlungen verändert. Dies ermöglicht Domänenexperten die manuelle Analyse der generierten Soll-Architektur und die Überprüfung ihrer potentiellen Restrukturierungsentscheidungen.

Zur Umsetzung dieses Simulationsansatzes wird die *Delta Modellierung* (vgl. Unterabschnitt 2.1.3) als transformierender Mechanismus für das 150%-Modell verwendet. Hierfür sind die einzelnen Deltaoperationen bereits für alle Restrukturierungsempfehlungen im Rahmen der Differenzanalyse bestimmt worden (vgl. Unterabschnitt 5.1.1). Diese sind für jede Restrukturierungsempfehlung zu einem Deltamodul zusammengefasst und an die jeweilige Empfehlung angehangen worden. So kann die Simulation von ausgewählten Restrukturierungsempfehlungen über die Anwendung ihrer spezifischen Deltamodule erfolgen, was zur Transformation des betrachteten 150%-Modells führt. Dieser Ansatz ist in Abbildung 5.9 grafisch dargestellt.

Um den Ansatz zur Simulation von Restrukturierungsempfehlungen besser zu verdeutlichen, wird unser fortlaufendes Beispiel wieder aufgegriffen. Die folgende Tabelle 5.3 zeigt dazu die `Applikation-Server-Intersection` als Ausschnitt aus dem bereits vorgestellten Ist-150%-Modell (vgl. Tabelle 3.12).

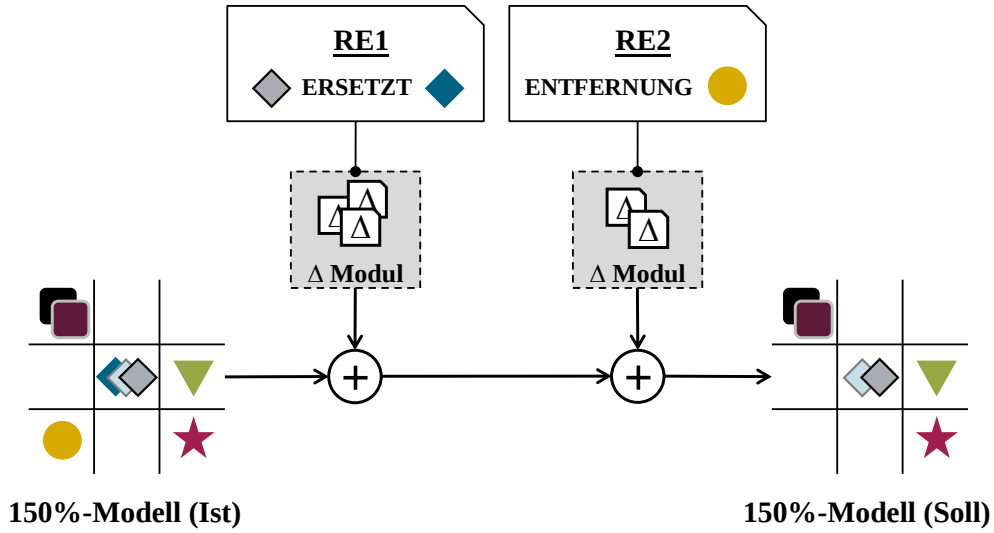


Abbildung 5.9: Simulation von ausgewählten Restrukturierungsempfehlungen

Wie in Tabelle 5.3 zu sehen ist, beinhaltet die `Applikation-Server-Intersection` neben den beiden optionalen Konfigurationsoptionen `O3 (OpenLDAP)` und `O4 (Tivoli Access Manager)` auch die Alternativgruppe `AG1` mit den beiden alternativen Konfigurationsoptionen `A1 (Tomcat)` und `A2 (WebSphere App. Server)`. Für diese Variabilitätsgruppe wurde in Kapitel 4 die Ersetzungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` identifiziert. Weiterhin sind für diese Empfehlung im Rahmen der Differenzanalyse (vgl. Unterabschnitt 5.1.1) alle erforderlichen Deltaoperationen bestimmt und dieser hinzugefügt worden. Das bereits vorgestellte Listing 5.2 enthält alle generierten Deltaoperationen für `RE1`.

Für die Simulation der Ersetzungsempfehlung `RE1` werden nun die spezifischen Deltaoperationen auf das Ist-150%-Modell angewendet, um dieses automatisiert zu trans-

	Server		
Applikation		A1 (3/4) Tomcat	7 (2/3)
	AG1		6 (1/3)
		A2 (1/4) WebSphere App. Server	7.0 (1/1)
		O3 (1/4) OpenLDAP	2.1 (1/1)
		O4 (1/4) Tivoli Access Manager	6.1 (1/1)

AG = Alternativgruppe *A* = alternativ *O* = optional

Tabelle 5.3: Applikation-Server-Intersection des Ist-150%-Modell aus dem fortlaufenden Beispiel

	Server		
Applikation	V1	(4/4) Tomcat	7 (3/4) 6 (1/4)
	O3	(1/4) OpenLDAP	2.1 (1/1)
	O4	(1/4) Tivoli Access Manager	6.1 (1/1)

V = verpflichtend O = optional

Tabelle 5.4: Applikation-Server-Intersection des Soll-150%-Modell für das fortlaufenden Beispiel

formieren. So wird beispielsweise die Komponente `WebSphere App. Server 7.0` mit `removeModelElementFromModel('WebSphere App. Server 7.0','TAP3')` aus dem `TAP3` entfernt und dafür die ersetzende Komponente `Tomcat 7` mit `addContainingModelToModelElement('TAP3','Tomcat 7')` diesem `TAP3` hinzugefügt. Nach Anwendung aller Deltaoperationen entsteht so als Ergebnis ein verändertes 150%-Modell, welches die gewünschte Soll-Architektur mit der resultierenden Variabilität nach Umsetzung der Restrukturierungsempfehlung repräsentiert.

Die folgende Tabelle 5.4 zeigt den betrachteten Ausschnitt des generierten Soll-150%-Modells für unser fortlaufendes Beispiel. Wie hier zu erkennen ist, kann durch die Umsetzung der Ersetzungsempfehlung `RE1` die Eliminierung der beiden Alternativen `A1` und `A2` sowie der zugehörigen Alternativgruppe `AG1` erreicht werden. Dies ist möglich, da die Komponente `Tomcat` nun in allen vier betrachteten TAPs verwendet wird und demnach als verpflichtend (`V1`) anzusehen ist. Solche Auswirkungen einer Restrukturierungsempfehlung auf die Variabilität von analysierten TAPs können mit Hilfe der Simulation leicht erkannt werden und so Domänenexperten bei der Entscheidungsfindung unterstützen.

5.3.3 Entscheidung zur Restrukturierung

Auf der Grundlage des erstellten Soll-150%-Modells ist es Domänenexperten möglich, eine Entscheidung zur Restrukturierung der analysierten TAPs zu treffen. Hierfür können sie das Ist-150%-Modell mit einem konkreten Soll-150%-Modell vergleichen, um die Auswirkungen von ausgewählten Restrukturierungsempfehlungen auf die Variabilität der betroffenen TAPs manuell zu untersuchen. Dadurch sind sie in der Lage, die Konsequenzen einer tatsächlichen Restrukturierung abzuschätzen und eine entsprechende Restrukturierungsentscheidung zu treffen.

Basierend auf solch einem Soll-150%-Modell haben Domänenexperten drei konkrete Entscheidungsalternativen, welche sich in den bereits vorgestellten iterativen Entscheidungsprozess einbetten (vgl. Abbildung 5.7). Diese drei Entscheidungsalternativen sind in Abbildung 5.10 graphisch dargestellt und werden nachfolgend näher erläutert.

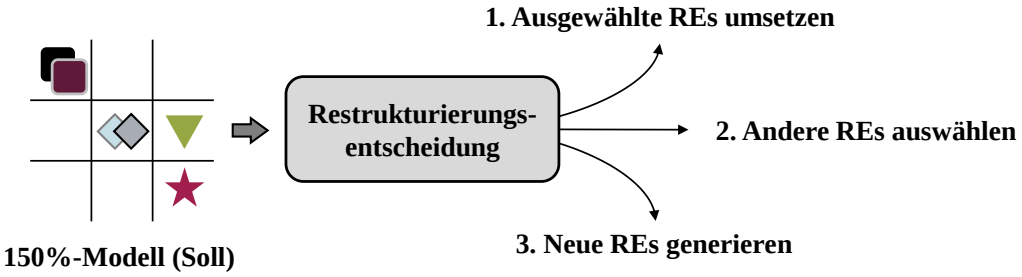


Abbildung 5.10: Entscheidungsalternativen für ausgewählte Restrukturierungsempfehlungen (REs)

1. Ausgewählte Restrukturierungsempfehlungen umsetzen: Im ersten und besten Fall bewerten Domänenexperten das resultierende Soll-150%-Modell als *geeignet* und die beabsichtigten Restrukturierungsempfehlungen als umsetzbar. Das bedeutet, dass die Auswirkungen und Konsequenzen beurteilt sind und einer erfolgreichen Restrukturierung nicht im Wege stehen. Somit kann die Entscheidung zugunsten einer Umsetzung der ausgewählten Restrukturierungsempfehlungen getroffen werden. Für diese Empfehlungen erfolgt im nächsten Schritt die *Restrukturierungsplanung* (vgl. Abschnitt 5.4).

2. Andere Restrukturierungsempfehlungen auswählen: Im zweiten Fall bewerten Domänenexperten das entsprechende Soll-150%-Modell und die zugrunde liegenden Restrukturierungsempfehlungen zumindest teilweise als *nicht geeignet*. Dies kann beispielsweise daran liegen, dass die Auswirkungen auf die Variabilität doch nicht beabsichtigt sind (z.B. Abschaltung aller alternativen Komponenten einer Gruppe) oder die Konsequenzen einer Restrukturierung als nicht erfolgversprechend betrachtet werden (z.B. gleichzeitige Anpassung zu vieler Softwaresysteme). In solch einer Situation kann das entsprechende Soll-150%-Modell einfach verworfen werden und eine *Auswahl anderer Restrukturierungsempfehlungen* erfolgen (vgl. Unterabschnitt 5.3.1), für die dann ein neues Soll-150%-Modell erzeugt wird.

3. Neue Restrukturierungsempfehlungen generieren: Im letzten Fall erkennen Domänenexperten weiteres Restrukturierungspotential für das erzeugte Soll-150%-Modell. Dies kann zum Beispiel eine zusätzliche Ersetzung von einer Komponente innerhalb einer Relationsgruppe sein. In solch einem Fall kann der iterative Entscheidungsprozess erneut durchlaufen werden, indem das Soll-150%-Modell als Input für die *Ableitung neuer Restrukturierungsempfehlungen* (vgl. Kapitel 4) verwendet wird. So können mehrere aufeinander aufbauende Soll-150%-Modelle erzeugt werden, sodass Domänenexperten die Möglichkeit haben, sich schrittweise ihrer optimale Lösung zu nähern. Dabei werden alle Restrukturierungsempfehlungen, welche für ein entsprechendes Soll-150%-Modell umzusetzen sind, zwischengespeichert. Mit jeden neuen Soll-150%-Modell kann eine Entscheidung auch zugunsten 1. oder 2. getrof-

fen werden. Im ersten Fall werden dann die Restrukturierungsempfehlungen aller Soll-150%-Modelle für die Umsetzung im nächsten Schritt *Restrukturierungsplanung* weiterverarbeitet. Im zweiten Fall werden die Restrukturierungsempfehlungen des zu verwerfenden Soll-150%-Modells gelöscht, sodass auf dem zuvor generierten Soll-150%-Modell weitergearbeitet werden kann und *andere Restrukturierungsempfehlungen* zur Simulation *ausgewählt* werden können.

5.4 Restrukturierungsplanung

Nachdem Domänenexperten die Entscheidung zur Restrukturierung der betrachteten TAPs getroffen und sich damit auf die Umsetzung ausgewählter Restrukturierungsempfehlungen festgelegt haben, kann nun die Planung der Restrukturierung erfolgen. Hierbei geht es vor allem darum, die erforderlichen *Migrations-* und *Anpassungsmaßnahmen* für jeden einzelnen TAP zu identifizieren und in eine geeignete Reihenfolge für die Umsetzung zu bringen. Daraus wird abschließend ein *Restrukturierungsplan* erzeugt, welcher alle wichtigen Informationen zur anstehenden Restrukturierung für Domänenexperten zur Verfügung stellt.

Eine *Migrationsmaßnahme (MM)* beinhaltet dabei alle Tätigkeiten, die mit der Restrukturierung eines zu ersetzenden oder zu entfernenden physischen Elementes `B` innerhalb eines spezifischen TAPs `j` verbunden sind. Hierzu wird für jeden TAP `j`, der von einer konkreten Restrukturierungsempfehlung `REi (A ERSETZT B)` betroffen ist, eine entsprechende Migrationsmaßnahme `MMi (A ERSETZT B, TAPj)` erzeugt. Zudem wird für die Realisierung einer Entfernungsempfehlung `REi (ENTFERNUNG B)` im TAP`j` die entsprechende Migrationsmaßnahme `MMi (ENTFERNUNG B, TAPj)` generiert. So lässt sich beispielsweise für unsere exemplarische Restrukturierungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)`, welche unter anderem die beiden Restrukturierungstätigkeiten *Installation* und *Konfiguration* der neuen Komponente `Tomcat 7` im TAP `3` beinhaltet, die konkrete Migrationsmaßnahme `MM1 (Tomcat 7 ERSETZT WebSphere App. Server 7.0, TAP3)` generieren.

Im Gegensatz dazu beschreibt eine *Anpassungsmaßnahme (AM)* alle Tätigkeiten, die mit der Anpassung einer physischen Komponente `C` im TAP `j` aufgrund einer Anpassungsabhängigkeit zur Komponente `B` (`B` $\rightarrow$ `C`) einer umzusetzenden Migrationsmaßnahme `MMi (A ERSETZT B, TAPj)` oder `MMi (ENTFERNUNG B, TAPj)` verbunden sind. So wird für jedes von `B` abhängige Element `C` im TAP `j` eine spezifische Anpassungsmaßnahme `AMk (ANPASSUNG C, MMi, TAPj)` generiert. Als Beispiel kann hier die Tätigkeit *Schnittstellenanpassung* bei der Komponente `IBM DB2 10` genannt werden, die aufgrund der identifizierten Anpassungsabhängigkeit `WebSphere App. Server` $\rightarrow$ `IBM DB2` (vgl. Unterabschnitt 5.1.2) für `MM1 (Tomcat 7 ERSETZT WebSphere App. Server 7.0, TAP3)` im TAP `3` zu berücksichtigen ist. Hierfür wird die konkrete Anpassungsmaßnahme `AM1 (ANPASSUNG IBM DB2 10, MM1, TAP3)` erzeugt.

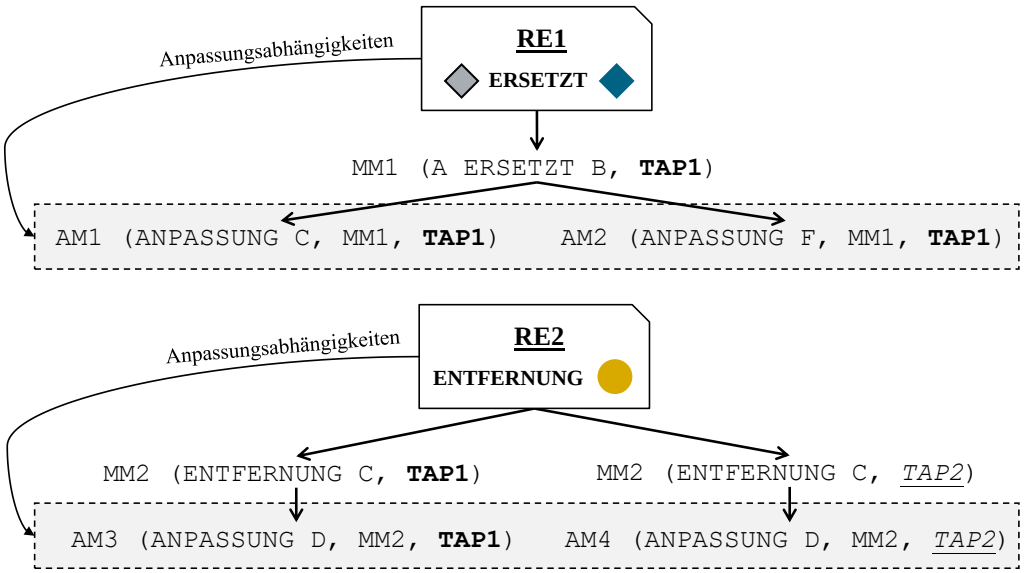


Abbildung 5.11: Generierung von Migrations- und Anpassungsmaßnahmen

So werden für die Restrukturierungsplanung zunächst alle konkreten Maßnahmen für jeden einzelnen der analysierten TAPs bestimmt. Zu diesem Zweck wird für jede Restrukturierungsempfehlung `REi` jeweils eine konkrete Migrationsmaßnahme `MMi` für jeden betroffenen TAP `j` erstellt. Anschließend können die erforderlichen Anpassungsmaßnahmen `AMk` für jede Migrationsmaßnahme `MMi` auf Grundlage der identifizierten und verifizierten Anpassungsabhängigkeiten von `REi` generiert werden. Dieses Vorgehen ist schematisch in Abbildung 5.11 dargestellt.

In unserem fortlaufenden Beispiel würde also für die Restrukturierungsempfehlung `RE1 (Tomcat ERSETZT WebSphere App. Server)` die konkrete Migrationsmaßnahme `MM1 (Tomcat 7 ERSETZT WebSphere App. Server 7.0, TAP3)` erzeugt werden. Da von dieser Restrukturierungsempfehlung lediglich der `TAP3` betroffen ist (vgl. Tabelle 3.1), wird keine weitere Migrationsmaßnahme benötigt. Im Rahmen der Anpassungsbedarfsanalyse sind für `RE1` die beiden anpassungsabhängigen Elemente `IBM DB2` und `Firefox` identifiziert worden (vgl. Unterabschnitt 5.1.2). Basierend darauf können somit die beiden erforderlichen Anpassungsmaßnahmen `AM1 (ANPASSUNG IBM DB2 10, MM1, TAP3)` und `AM2 (ANPASSUNG Firefox 49, MM1, TAP3)` generiert werden.

Nachdem alle konkreten Migrations- und Anpassungsmaßnahmen identifiziert wurden, kann ihre *Implementierungsreihenfolge* bestimmt werden. Hierbei wird für jeden TAP `j` definiert, in welcher Reihenfolge die einzelnen Maßnahmen, die diesen TAP `j` betreffen, umgesetzt werden sollen. Dies hängt dabei von der gewählten *Implementierungsstrategie* und den vorhandenen *Implementierungsabhängigkeiten* ab.

Die Implementierungsstrategie definiert eine *bevorzugte* Umsetzungsreihenfolge innerhalb eines konkreten TAPs. Diese ist von Domänenexperten nach unterneh-

mensindividuellen Präferenzen zu wählen und kann entweder *horizontal* oder *vertikal* ausgelegt sein. Erstere bewirkt die aufeinanderfolgende Umsetzung von Maßnahmen, die sich auf dem selben Layer befinden. So werden alle Maßnahmen für den gleichen Layer zusammen umgesetzt, auch wenn sie unterschiedliche Tiers betreffen. Als Beispiel hierfür können Anpassungen am Layer `Betriebssystem` genannt werden, die zuerst für den `Client-` und den `Server-Tier` realisiert werden, bevor eine weitere Anpassung auf dem Layer `Präsentation` erfolgen kann. Im umgekehrten Fall wird von einer vertikalen Implementierungsstrategie gesprochen, wenn anstehende Maßnahmen immer gesammelt für einen spezifischen Tier umgesetzt werden. So würden beispielsweise Anpassungen auf den Layern `Betriebssystem` und `Präsentation` zuerst auf dem `Client-Tier` erfolgen, bevor eine Anpassung auf dem `Betriebssystem`-Layer des `Server-Tiers` durchgeführt werden kann.

Darüber hinaus beschreiben Implementierungsabhängigkeiten eine *erforderliche* Umsetzungsreihenfolge für Migrations- und Anpassungsmaßnahmen innerhalb eines spezifischen TAPs `j`. Wie in Abbildung 5.12 zu erkennen ist, gibt es dabei drei unterschiedliche Typen (*A, B, C*) von Implementierungsabhängigkeiten.

Der *Typ A* beschreibt eine Implementierungsabhängigkeit zwischen zwei Migrationsmaßnahmen `MM1` und `MM2` mit demselben ersetzenden Element `A`. Diese sollten direkt nacheinander umgesetzt werden, um ähnlichen Maßnahmen zu bündeln und dadurch den Migrationsaufwand zu reduzieren. Dabei hat die konkrete Implementierungsreihenfolge von `MM1` und `MM2` keine Relevanz.

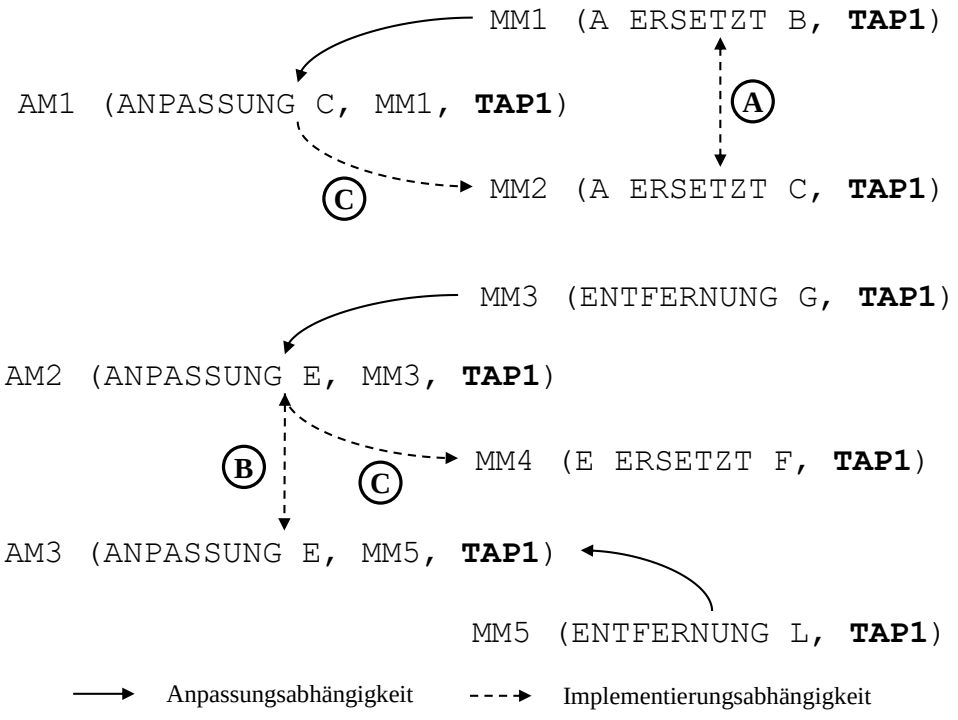


Abbildung 5.12: Drei Typen von Implementierungsabhängigkeiten

Der *Typ B* entspricht einer Implementierungsabhängigkeit zwischen zwei Anpassungsmaßnahmen `AM2` und `AM3` mit dem gleichen anzupassenden Element `E`. Auch diese beiden Anpassungen am selben Element sollten zusammen durchgeführt werden, um den Anpassungsaufwand dafür so gering wie möglich zu halten. Eine konkrete Reihenfolge ist aber auch bei der Umsetzung dieser beiden Maßnahmen nicht von Bedeutung.

Der dritte *Typ C* repräsentiert eine Implementierungsabhängigkeit zwischen einer Migrationsmaßnahme und einer Anpassungsmaßnahmen. Dabei kann entweder eine Anpassungsmaßnahme `AM2` ein ersetzendes Element `E` einer Migrationsmaßnahme `MM4` betreffen oder eine Anpassungsmaßnahme `AM1` ein zu entfernendes Element `C` einer Migrationsmaßnahme `MM2`. In beiden Fällen muss die jeweilige Migrationsmaßnahme zuerst durchgeführt werden, da zum einen die Anpassungsmaßnahme `AM1` nach Eliminierung des Elementes `C` durch `MM2` nicht mehr erforderlich ist und zum anderen die Anpassung vom Element `E` durch `AM2` nach Durchführung von `MM4` für das dann ersetzte Element `F` wiederholt werden müsste.

Zur Bestimmung der Implementierungsreihenfolge für die generierten Migrations- und Anpassungsmaßnahmen eines spezifischen TAPs `j` werden die vorhandenen Implementierungsabhängigkeiten und die gewählte Implementierungsstrategie berücksichtigt. Zu diesem Zweck erfolgt zuerst die Identifizierung von Implementierungsabhängigkeiten zwischen den einzelnen Maßnahmen eines konkreten TAPs `j`, indem diese im Hinblick auf die vorgestellten drei Abhängigkeitstypen A, B und C analysiert werden. Kann dabei eine konkrete Implementierungsabhängigkeit zwischen zwei beabsichtigten Maßnahmen ermittelt werden, wird ein spezifisches Set gebildet, in welches diese beiden Maßnahmen eingeordnet werden. Können anschließend weitere Maßnahmen identifiziert werden, die ebenfalls eine Implementierungsabhängigkeit zu den bereits zugeordneten Maßnahmen aufweisen, so werden diese in das gleiche Set einsortiert. Darüber hinaus werden einem konkreten Set auch die jeweiligen Anpassungsmaßnahmen zugeordnet, die aufgrund einer spezifischen Anpassungsabhängigkeit zu einer Migrationsmaßnahme des betreffenden Sets erforderlich sind. So entstehen im Verlauf dieser Abhängigkeitsanalyse mehrere Sets von Maßnahmen, die durch eine Implementierungsabhängigkeit oder Anpassungsabhängigkeit in Verbindung zueinander stehen und zusammenhängend umgesetzt werden sollten. Die nachfolgende Abbildung 5.13 zeigt ein Beispiel mit zwei identifizierten Sets, die aufgrund der Implementierungs- und Anpassungsabhängigkeiten aus der Abbildung 5.12 erstellt werden können.

Auf Basis solcher generierten Sets kann anschließend die konkrete Implementierungsreihenfolge bestimmt werden. Hierbei wird zwischen der *internen* und der *externen* Reihenfolge unterschieden. Die interne Reihenfolge definiert die Umsetzung innerhalb eines spezifischen Sets. Diese ist dadurch bestimmt, dass zuerst alle Migrationsmaßnahmen eines Sets implementiert werden, bevor die entsprechenden Anpassungsmaßnahmen folgen. So können etwaige Implementierungsabhängigkeit vom Typ C aufgelöst werden, indem beispielsweise nicht mehr erforderliche Anpassungsmaßnahmen

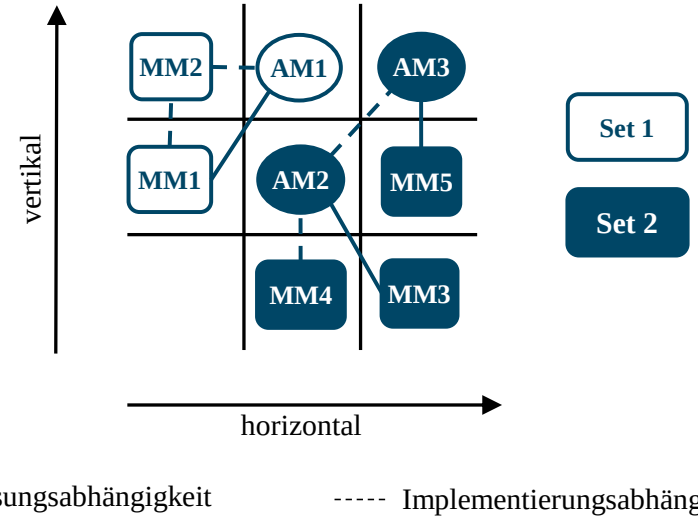


Abbildung 5.13: Zwei Sets mit Migrations- und Anpassungsmaßnahmen

erkannt und gelöscht werden. Innerhalb eines Sets kann dabei eine beliebige Migrationsmaßnahme als Startpunkt gewählt werden. Demnach würden beispielsweise für das `Set 1` in der Abbildung 5.13 zuerst die Migrationsmaßnahmen `MM1` und `MM2` realisiert werden, bevor die Anpassungsmaßnahme `AM1` umgesetzt wird.

Nach Festlegung der internen Umsetzung aller identifizierten Sets kann die externe Implementierungsreihenfolge bestimmt werden, indem diese Sets anhand der ausgewählten Implementierungsstrategie geordnet werden. Hierfür sind die jeweiligen Layer und Tiers der zu realisierenden Migrationsmaßnahmen in den einzelnen Sets zu berücksichtigen. Bei einer vertikalen Implementierungsstrategie werden solche Sets zuerst umgesetzt, die Migrationsmaßnahmen auf dem untersten Layer beinhalten. Anschließend werden die Sets mit Migrationsmaßnahmen auf darüber liegenden Layern durchgeführt. Ähnlich verhält es sich bei einer horizontalen Implementierungsstrategie, bei der die Sets nach den Tiers von links nach rechts sortiert werden. So kann die externe Reihenfolge der jeweiligen Sets auf Basis der Implementierungsstrategie bestimmt werden. Aus der schematischen Darstellung in Abbildung 5.13 geht beispielsweise hervor, dass bei einer vertikalen Implementierungsstrategie zuerst das `Set 2` und anschließend das `Set 1` realisiert wird. Bei einer horizontalen Implementierungsstrategie würde die Umsetzung in umgekehrter Reihenfolge durchgeführt werden, also zuerst für das `Set 1` und danach für das `Set 2`.

Als Ergebnis dieser Analyse wird ein *Restrukturierungsplan* für Domänenexperten erzeugt, welcher die ermittelte Implementierungsreihenfolge aller Migrations- und Anpassungsmaßnahmen für die jeweiligen TAPs anschaulich darstellt. Solch ein exemplarischer Restrukturierungsplan ist für unser fortlaufendes Beispiel in der Tabelle 5.5 zu sehen. Wie hier zu erkennen ist, enthält der Restrukturierungsplan Informationen über die betroffenen Softwaresysteme (TAPs), die ausgewählte Implementierungsstrategie und die zur Umsetzung bestimmten

Restrukturierungsplan 1	
Betroffene Systeme:	TAP1, TAP3, TAP4
Implementierung:	vertikal (↑)
Ausgewählte Restrukturierungsempfehlungen:	`RE1 (Tomcat ERSETZT WebSphere App. Server)` `RE2 (ENTFERNUNG phpMyAdmin)`
System **TAP1**:	`MM2 (ENTFERNUNG phpMyAdmin 4.6)` → `AM3 (ANPASSUNG MySQL 5.6)`
System **TAP3**:	`MM1 (Tomcat 7 ERSETZT WebSphere App. Server 7.0)` → `AM1 (ANPASSUNG IBM DB2 10)` → `AM2 (ANPASSUNG Firefox 49)`
System **TAP4**:	`MM2 (ENTFERNUNG phpMyAdmin 2.5)` → `AM4 (ANPASSUNG MySQL 5.7)`

Tabelle 5.5: Restrukturierungsplan für das fortlaufenden Beispiel

Restrukturierungsempfehlungen. Hierfür werden alle Migrations- und Anpassungsmaßnahmen für jeden zu restrukturierenden TAP einzeln und sortiert nach der ermittelten Implementierungsreihenfolge aufgelistet. Dabei symbolisiert das Zeichen '→' vor einer spezifischen Anpassungsmaßnahme, dass diese aufgrund ihrer Anpassungsabhängigkeit zur darüber liegenden Migrationsmaßnahme gehört. Hierdurch wird die identifizierte interne Reihenfolge beschrieben. So ist beispielsweise für den `TAP3` die entsprechende Migrationsmaßnahme `MM1 (Tomcat 7 ERSETZT WebSphere App. Server 7.0, TAP3)` aufgeführt, welche die beiden Anpassungsmaßnahmen `AM1 (ANPASSUNG IBM DB2 10, MM1, TAP3)` und `AM2 (ANPASSUNG Firefox 49, MM1, TAP3)` nach sich zieht. Durch die Sortierung der einzelnen Zeilen mit den jeweiligen TAPs wird die externe Reihenfolge dargestellt. So sind in diesem Beispiel die Maßnahmen des `TAP1` vor den entsprechenden Maßnahmen für den `TAP3` und `TAP4` umzusetzen.

Solch ein Restrukturierungsplan liefert eine gute Übersicht über die beabsichtigten Restrukturierungen in Form von erforderlichen Migrations- und Anpassungsmaßnahmen für die betroffenen Softwaresysteme. Anhand dessen ist es Domänenexperten möglich, konkrete Projekte für die Umsetzung dieser Restrukturierungsmaßnahmen zu initiieren und dadurch die unnötige Variabilität in den analysierten TAPs nachhaltig zu reduzieren.

5.5 Verwandte Arbeiten

Im folgenden Abschnitt werden die verwandten Arbeiten vorgestellt, welche im Kontext dieses Kapitel relevant sind. Dabei liegt der Fokus auf solchen Arbeiten, die entweder Methoden zur Bewertung einer Technologie- beziehungsweise einer Unternehmensarchitektur bieten oder geeignete Ansätze zur Unterstützung von Architekturentscheidungen liefern.

Bewertungsmethoden für Unternehmensarchitekturen

Die Bewertung von Unternehmensarchitekturen kann aus unterschiedlichen Blickwinkeln erfolgen. Hierzu zählen beispielsweise die *Wartung*, die *Integration*, die *Planung* und das *Management* solcher Architekturen. Eine gute Übersicht über entsprechende Bewertungsansätze für diese und weitere Bewertungsperspektiven liefern **Nikpay et al.** [NARS16]. Im Kontext dieses Kapitels spielen vor allem die folgenden Arbeiten eine Rolle, da sie Methoden für die Bewertung von Artefakten im Rahmen der Wartung und Weiterentwicklung von Technologie- beziehungsweise Unternehmensarchitekturen vorschlagen.

Österlind et al. [OJK+13] präsentieren eine Methode zur Bewertung von Unternehmensarchitekturen anhand von Qualitätsmerkmalen[3] wie beispielsweise Kosten, Verfügbarkeit und Wiederverwendbarkeit. Hierzu wenden sie die *Nutzentheorie* an, welche einen Ansatz zur nummerischen Repräsentation von Stakeholder Präferenzen bietet. Durch die Berechnung von Nutzwerten kann das Architekturszenario bestimmt werden, welches auf Basis der definierten Präferenzen am geeignetsten ist.

Karimi et al. [KSD14] schlagen in ihrer Arbeit einen genetischen Algorithmus vor, der die Bewertung verschiedener Szenarien einer Unternehmensarchitektur im Hinblick auf ihre Qualitätsmerkmale erlaubt. Dieser Algorithmus baut dabei auf der generalisierten *Architecture Tradeoff Analysis Method (ATAM)* nach Kazman et al. [KKC00] auf, welche eine Methode zur Evaluierung von Unternehmensarchitekturen mit Hilfe von Qualitätsszenarien darstellt. Durch die Anwendung des genetischen Algorithmus können diese Qualitätsszenarien priorisiert werden und unterstützen dadurch die Auswahl eines geeigneten Zielszenarios für die Unternehmensarchitektur.

Eskandari et al. [EN16] führen die Arbeit von Karimi et al. [KSD14] weiter aus und präzisieren den bereits vorgestellten Ansatz zur Priorisierung von Qualitätsszenarien in Unternehmensarchitekturen durch die Verwendung des *Non-Dominated Sorting Genetic Algorithm (NSGA-II)*. Hierdurch ist es ihnen möglich, die Geschwindigkeit des bisherigen Ansatzes zu verbessern und weitere Kriterien für die Priorisierung zu berücksichtigen.

Razavi et al. [RS09, RS12] präsentieren einen weiteren Ansatz zur Analyse von Architekturszenarien auf Basis der Qualitätsmerkmale. Hierzu schlagen sie eine quantitative Bewertungsmethode vor, welche die Erreichung ausgewählter Qualitätsziele

[3] Qualitätsmerkmale für Unternehmensarchitekturen sind beispielsweise in [Kha11] beschrieben

für jedes Szenario durch die Verwendung des *Analytical Hierarchy Process (AHP)* in Kombination mit dem Wissen und der Erfahrung von Domänenexperten ermittelt.

Lagerström et al. [Lag07, LJ08, LFJU09, LJH10] haben in ihren Arbeiten Ansätze zur Analyse von Unternehmensarchitekturen vorgestellt, mit deren Hilfe die Wartbarkeit und die Modifizierbarkeit solcher Architekturen bewertet werden können. Zur Analyse und Bewertung der Wartbarkeit präsentieren sie geeignete Modelle der Unternehmensarchitektur und aussagekräftige Einflussdiagramme für die graphische Darstellung. Im Hinblick auf die Modifizierbarkeit einer Unternehmensarchitektur präsentieren sie ein weiteres Vorgehen auf Basis der erstellten Architekturmodelle. Diese werden in Form von *Probabilistic Relational Models (PRMs)* formalisiert, sodass auch semantische Aspekte für eine Analyse von Unternehmensarchitekturen unter Unsicherheit berücksichtigt werden können. Um Domänenexperten bei der Anwendung dieser Methoden zu unterstützen, sind zudem geeignete Tools vorgestellt worden [JJSU07, EFJ+09, BUF+10].

Busch et al. [BZ18] präsentieren in ihrer Arbeit die *Enterprise Architecture Modifiability Analysis Method (EAMAM)*. Diese stellt einen weiteren Ansatz zur Analyse der Modifizierbarkeit von Unternehmensarchitekturen auf Basis der *Software Architecture Analysis Method (SAAM)* nach Kazman et al. [KWAB94] dar. Die vorgeschlagene Methode EAMAM bietet dabei einen Bewertungsprozess für verschiedene Szenarien von Unternehmensarchitekturen auf Basis von Analysemethoden (z.B. eine Change-Impact-Analyse) und Bewertungstechniken (z.B. die Identifizierung von Anforderungen für mögliche Restrukturierungen).

Die hier vorgestellten Arbeiten liefern zwar gute Ansätze zur Bewertung von Unternehmensarchitekturen, welche auch für Technologiearchitekturen anwendbar sind, jedoch ermöglichen sie nicht die Bewertungen von konkreten Restrukturierungsempfehlungen in Bezug auf ihre Variabilitätsdifferenzen, Anpassungsabhängigkeiten und erforderlichen Aufwände. Der Grund dafür liegt vor allem darin, dass keiner dieser Ansätze auf die Reduzierung von Variabilität in Technologie- beziehungsweise Unternehmensarchitekturen fokussiert ist. Vielmehr steht bei den gezeigten Ansätzen die Bewertung solcher Architekturen mit dem Ziel der Qualitätsverbesserung (z.B. [OJK+13, KSD14, EN16]) oder der Unterstützung von allgemeinen Restrukturierungen (z.B. [LJH10, BZ18]) im Vordergrund.

Letztere ließen sich dabei auch für vereinzelte Analyseschritte aus unserem Ansatz verwenden. So könnte beispielsweise die Change-Impact-Analyse aus dem Kontext von EAMAM [BZ18] für unsere Anpassungsbedarfsanalyse (vgl. Unterabschnitt 5.1.2) eingesetzt werden. Zudem wäre die Analyse der Wartbarkeit in Bezug auf die voraussichtlichen Kosten nach Lagerström et al. [LJH10] für unsere Aufwandsschätzung (vgl. Unterabschnitt 5.1.3) nutzbar. Allerdings bieten diese Ansätze jeweils nur einzelne Methoden für eine teilweise Bewertung von Restrukturierungsempfehlungen an. So ließe sich damit beispielsweise nicht die Variabilitätsdifferenz und damit das spezifische Variabilitätsreduktionspotential solcher Empfehlungen bestimmen. Dies ist jedoch für Domänenexperten im Rahmen der Entscheidungsfindung relevant. Zudem erfordern diese Ansätze einen hohen manuellen Aufwand für

die Erfassung der erforderlichen Daten (z.B. von konkreten Abhängigkeiten und spezifischen Kosten) durch Domänenexperten, wohingegen unser Ansatz automatisierte Methoden mit einer generischen Herangehensweise (z.B. mit Hilfe des Abhängigkeitsmodells und der Aufwandsmatrix) bereitstellt. So ermöglichen wir eine effiziente Analyse von großen Technologiearchitekturen in gewachsenen IT-Landschaften.

Entscheidungsunterstützung für Unternehmensarchitekturen

Die folgenden Arbeiten stellen Ansätze vor, die Domänenexperten bei der Entscheidungsfindung im Hinblick auf die Restrukturierung von Technologie- beziehungsweise Unternehmensarchitekturen unterstützen können.

Jugel et al. [JSZ15, JKSZ15a, JKSZ15b] präsentieren in ihren Arbeiten einen Ansatz zur Entscheidungsunterstützung basierend auf einem annotierten Metamodell. Dieses umfasst sowohl entscheidungsrelevante Informationen sowie Daten über betroffene Elemente einer Unternehmensarchitektur. Architekturmodelle, die diesem Metamodell entsprechen, können im Rahmen eines Entscheidungsprozesses, welcher die *Adaptive Case Management (ACM)-Methode* berücksichtigt, durch Analysetechniken und interaktive Visualisierungen inspiziert werden. Hierdurch können Domänenexperten bei der Findung von konkreten Architekturentscheidungen unterstützt werden.

Javanbakht et al. [JRSS08, JPD09] stellen in ihrem Ansatz eine Methode vor, welche Experten dabei unterstützen soll, eine Entscheidung über die Weiterentwicklung oder die Restrukturierung einer bestehenden Unternehmensarchitektur treffen zu können. Hierfür schlagen sie eine praxisnahe und multikriterielle Bewertungsmethode vor, welche beispielsweise die organisationalen Ziele und Beschränkungen sowie die unternehmensspezifische Mission berücksichtigt. Durch die Gewichtung der einzelnen Kriterien kann die vorgeschlagene Bewertungsmethode dann dafür verwendet werden, eine Entscheidung für oder gegen eine Restrukturierung abzuleiten.

Plataniotis et al. [PdL+13, PdP13, PdP14, vvMP14] präsentieren den *EA-Anamnesis*-Ansatz zur Unterstützung von Domänenexperten bei der Entscheidungsfindung für Rationalisierungsmaßnahmen in Unternehmensarchitekturen. Hierfür haben sie ein Metamodell entwickelt, welches Designkriterien für Artefakte einer Unternehmensarchitektur abbildet. Aufbauend darauf kann ein sogenannter *Decision Design Graph* erstellt werden, mit dem Architekturentscheidungen im Hinblick auf ihre Auswirkungen und ihre Beziehungen zu einander untersucht werden können. Mit Hilfe von darauf abgestimmten ex-ante und ex-post Sichten lässt sich für Domänenexperten so herausfinden, welche potentiellen Konsequenzen die jeweiligen Entscheidungen nach einer Rationalisierung mit sich bringen.

Nakakawa et al. [NvP11] schlagen in ihrer Arbeit einen Ansatz vor, mit dem eine kollektive Entscheidung mehrerer Domänenexperten im Hinblick auf Design-Alternativen für Unternehmensarchitekturen unterstützt wird. Zu diesem Zweck führen die Autoren die sogenannte *Collaborative Evaluation of Enterprise Architecture Design Alternatives (CEADA)*-Methode für die kollaborative Entscheidungsfindung ein. Im Kern dieser Methode

befindet sich ein Entscheidungsprozess, der es Domänenexperten ermöglicht, geeignete Design-Entscheidungen für eine effiziente Unternehmensarchitektur gemeinsam zu treffen.

Johnson et al. [JESP04, SLJ+05] stellen einen weiteren Ansatz zur Entscheidungsunterstützung vor, der eine kosteneffektive Methode für die Bewertung von unterschiedlichen Szenarien einer Unternehmensarchitektur beinhaltet. Solche Szenarien werden auf Basis verschiedener quantifizierbarer Systemeigenschaften (z.B. Modifizierbarkeit, Effizienz und Qualität) analysiert und miteinander verglichen. Die Ergebnisse werden anschließend in einer für Domänenexperten verständlichen Form grafisch dargestellt, sodass diese eine geeignete Grundlage für die Entscheidungsfindung bieten.

Lytra et al. [LZ13] präsentieren einen semi-automatisierten Ansatz zur Unterstützung von Domänenexperten im Rahmen der Bestimmung von *Architectural Design Decisions (ADDs)* unter Unsicherheit. Zu diesem Zweck haben sie ein auf der Fuzzylogik basierendes Expertensystem entwickelt, welches Design-Entscheidungen dokumentiert und deren mögliche Wiederverwendung analysiert. Durch eine eigens dafür erstellte DSL können Domänenexperten verschiedene Fuzzy-Entscheidungsmodelle spezifizieren, welche dann durch das Fuzzy-Inferenzsystem analysiert werden, um die bestmögliche Designlösung abzuleiten.

Chuang et al. [CvL10] schlagen einen weiteren Ansatz zur Entscheidungsunterstützung mit Hilfe der *Service-Dominant-Logic* vor, welche die Integration von *Business Values* in Architekturentscheidungen für Szenarien von Unternehmensarchitekturen ermöglicht. Nach den Autoren können Domänenexperten in Zusammenarbeit mit Kunden der jeweiligen IT-Systeme einen kontinuierlichen *Business-Value-Integrationsprozess* realisieren, um so auf Basis solcher Business Values und unter Berücksichtigung der jeweiligen Kundenerfahrungen sowie ihrem spezifischen Expertenwissen geeignete Architekturentscheidungen für die entsprechenden Szenarien treffen zu können.

All diese verwandten Arbeiten schlagen unterschiedliche Methoden vor, welche Domänenexperten bei der Findung von Architekturentscheidungen im Hinblick auf die Restrukturierung von Technologiearchitekturen unterstützen können. Dabei werden sowohl manuelle Unterstützungsmethoden (basierend auf Prozessen oder Metamodellen (z.B. [JSZ15, JRSS08, NvP11, CvL10])) als auch semi-automatisierte Ansätze (z.B. [PdP14, LZ13]) vorgeschlagen. Im Gegensatz zu diesen manuellen Methoden bietet unser Ansatz neben einem Metamodell für TAPs und einem iterativen Entscheidungsprozess auch automatisierte Techniken an, welche die Bewertung und Simulation von konkreten Restrukturierungsempfehlungen erlauben. Hierdurch sind Domänenexperten in der Lage, eine geeignete Entscheidung zur Restrukturierung der analysierten TAPs auf Basis einer automatisiert erstellten Entscheidungsgrundlage zu treffen.

Darüber hinaus unterstützen die verwandten semi-automatisierten Ansätze eher bei der Findung einer bestmöglichen Entscheidung unter verschiedenen Design-Alternativen. Hierzu bieten diese Ansätze beispielsweise geeignete Entscheidungsmo-

dellen (z.B. [LZ13]) oder Entscheidungsgraphen (z.B. [PdP14]) an. Dagegen erlaubt unser Ansatz, eine Auswahl mehrerer, möglicher Restrukturierungsempfehlungen zu simulieren, um so die resultierende Soll-Architektur sichtbar zu machen. Dadurch werden Domänenexperten in die Lage versetzt, die Auswirkungen von simulierten Restrukturierungsempfehlungen zu analysieren und dadurch nachhaltige Entscheidungen zur Reduzierung von unnötiger Variabilität zu treffen. So kann die Entscheidungsfindung im Kontext von großen, gewachsenen Technologiearchitekturen effektiv unterstützt werden.

5.6 Zusammenfassung

Die Bewertung von Restrukturierungsempfehlungen und die anschließende Entscheidungsfindung sind bisher mit hohem manuellen Aufwand verbunden. So stellt die Reduzierung von unnötiger Variabilität Domänenexperten regelmäßig vor eine große Herausforderung, da Entscheidungen für konkrete Restrukturierungen die intensive Analyse ihrer Auswirkungen in den betroffenen TAPs erfordern. Solche Analysen sind insbesondere für gewachsene IT-Landschaften mit hunderten oder tausenden von Softwaresystemen manuell kaum noch durchführbar.

Zur Lösung dieses Problems wurde in diesem Kapitel ein automatisierter Ansatz vorgestellt, der Domänenexperten bei der Bewertung von Restrukturierungsempfehlungen und der anschließenden Entscheidungsfindung für die Restrukturierung unterstützt. Zu diesem Zweck wurden zuerst im Abschnitt 5.1 Methoden zur Einzelbewertung von Restrukturierungsempfehlungen vorgestellt, mit deren Hilfe das Potential für die Variabilitätsreduzierung, die erforderlichen Anpassungen und die damit verbundenen Aufwände für jede einzelne Restrukturierungsempfehlung bestimmt werden können. Danach sind im Abschnitt 5.2 Methoden zur Bewertung des Gesamtportfolios von abgeleiteten Restrukturierungsempfehlungen präsentiert worden, die als Ergebnis identifizierte Korrelationen, eine Heatmap und eine Aufwand-Nutzen-Matrix für das Set aller Restrukturierungsempfehlungen zur Verfügung stellen.

Im nächsten Abschnitt 5.3 wurde anschließend gezeigt, wie konkrete Restrukturierungsentscheidungen auf Basis der Ergebnisse aus den beiden vorherigen Abschnitten getroffen werden können. Hierfür wurde ein iterativer Entscheidungsprozess vorgestellt, im Rahmen dessen Domänenexperten zuerst geeignete Restrukturierungsempfehlungen auswählen können, um diese anschließend zu simulieren. Hierdurch wird ein 150%-Modell der resultierenden Soll-Architektur erzeugt, welches die entstehende Variabilität abbildet. Auf dieser Grundlage kann dann eine Entscheidung zur Umsetzung der ausgewählten Restrukturierungsempfehlungen oder zur Auswahl anderer Empfehlungen für die Simulation getroffen werden. Aber auch eine Entscheidung zur Generierung neuer Restrukturierungsempfehlungen auf Basis des erstellten Soll-150%-Modells ist möglich. So kann eine neue Iteration des Entscheidungsprozesses angestoßen werden, was eine schrittweise Annäherung an die gewünschte Zielarchitektur ermöglicht.

Ist eine Entscheidung zugunsten der Umsetzung von ausgewählten und simulierten Restrukturierungsempfehlungen gefällt worden, wird ein geeigneter Restrukturierungsplan erzeugt. Wie im Abschnitt 5.4 präsentiert, werden zu diesem Zweck alle erforderlichen Migrations- und Anpassungsmaßnahmen für jeden einzelnen TAP aus den entsprechenden Restrukturierungsempfehlungen abgeleitet und in eine geeignete Implementierungsreihenfolge unter Berücksichtigung von Implementierungsabhängigkeiten und der gewählten Implementierungsstrategie gebracht. Als Ergebnis wird hieraus ein konkreter Restrukturierungsplan erzeugt, der Domänenexperten einen guten Überblick über alle beabsichtigten Maßnahmen in den betroffenen Softwaresystemen liefert und so als Grundlage für die Initiierung von konkreten Umsetzungsprojekten dienen kann.

6 Evaluation

In diesem Kapitel wird die Evaluation für unseren entwickelten Ansatz vorgestellt. Hierfür erläutert Abschnitt 6.1 zunächst die Implementierung des Softwareprototypen, bevor die durchgeführten Fallstudien in Abschnitt 6.2 und die absolvierten Experteninterviews in Abschnitt 6.3 präsentiert werden. Abschließend wird in Abschnitt 6.4 noch die Validität der Evaluationsergebnisse diskutiert.

6.1 Implementierung

Für die Evaluation unseres Ansatzes haben wir einen Softwareprototypen entworfen, der in diesem Abschnitt kurz vorgestellt wird. Im Rahmen der Entwicklung dieser prototypischen Implementierung wurden dabei die folgenden Ziele für den Architekturentwurf definiert:

1. Fokus durch *Kapselung* von Funktionalität
2. Einfache *Wiederverwendung* von Funktionalität
3. Leichte *Erweiterbarkeit* um zusätzliche Funktionalität

Durch die Beachtung dieser Architekturziele ist es möglich, die vorgestellten Methoden unseres Ansatzes mit Hilfe einer geeigneten Struktur umzusetzen und die Integration zukünftiger Weiterentwicklungen bereits von Anfang an zu berücksichtigen. Zur Realisierung dieser Ziele eignet sich beispielsweise der Architekturstil *Schichtenarchitektur* [BHS07]. Eine Architektur nach solch einem Stil ist modular aufgebaut und besteht aus aufeinander aufbauenden *Schichten*, auch als *Layer* bezeichnet, welche ein spezifisches Anwendungssystem nach Gruppen von zusammenhängenden Aufgaben zerlegen. Häufig werden solche Anwendungen dabei in die drei Schichten *Daten*, *Logik* und *Präsentation* unterteilt.

Um solch eine Schichtenarchitektur für unseren Softwareprototypen umzusetzen, wurde dieser auf Basis der *Eclipse Rich Client Platform (RCP)*[1] in der Programmiersprache *Java* implementiert. Die Eclipse RCP basiert dabei auf Schichten aus *Java-Plugins*, die einzelne Softwarekomponenten darstellen, in denen ihre Funktionalität gekapselt wird. Schnittstellen (*Extension-Points*) und Beziehungen (*Extensions*) der einzelnen Plugins definieren dabei, welche Abhängigkeiten zwischen den jeweiligen

[1] https://wiki.eclipse.org/Rich_Client_Platform

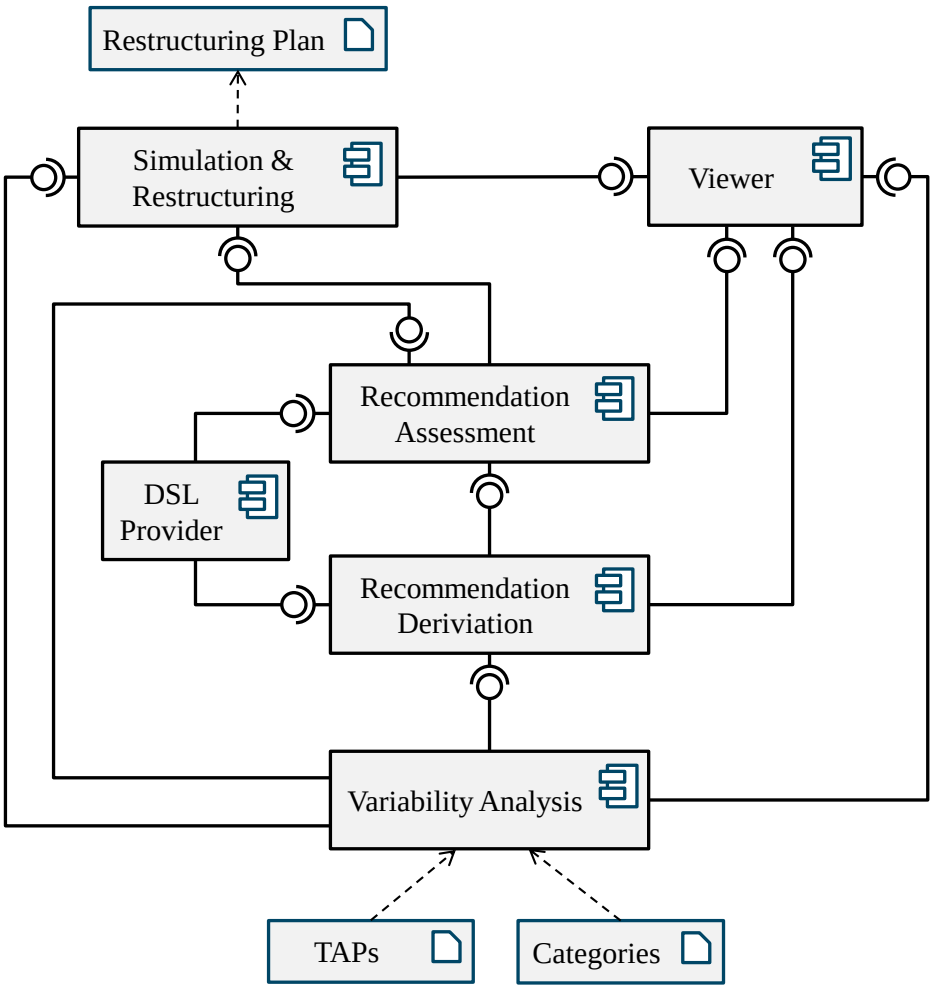


Abbildung 6.1: Architektur der prototypischen Implementierung

Plugins bestehen und wie Daten und Informationen ausgetauscht werden. Die Plugins für unseren Softwareprototypen haben wir mit Hilfe des *Eclipse Integrated Development Environment (IDE)*[2] entwickelt und zu einer gemeinsamen RCP-Applikation zusammengeführt.

Die Abbildung 6.1 zeigt die Architektur unseres Softwareprototypens anhand eines *Komponentendiagramms*. Darin sind verschiedene Oberkomponenten mit deren Schnittstellen und Beziehungen abgebildet. Diese Oberkomponenten enthalten weitere Unterkomponenten, welche einzelne Plugins repräsentieren, die für die prototypische Implementierung realisiert worden sind. Aus Gründen der Übersichtlichkeit wird in diesem Abschnitt nur die Architektur anhand ihrer Oberkomponenten näher erläutert. Detaillierte Informationen zu den einzelnen Unterkomponenten und deren Abhängigkeiten können dem Anhang entnommen werden (vgl. Abschnitt A.3).

[2]https://www.eclipse.org/ide/

Wie in Abbildung 6.1 zu erkennen ist, besteht die Architektur unseres Softwareprototypen aus den sechs Oberkomponenten *Variability Analysis*, *Recommendation Deriviation*, *Recommendation Assessment*, *DSL Provider*, *Simulation & Restructuring* sowie *Viewer*. Als Input werden zwei separate CSV-Dateien erwartet, welche zum einen die zu analysierenden *TAPs* und zum anderen deren *Kategorien* oder ihr *Kategoriensystem* beschreiben. Dabei hat sich die Verwendung von CSV-Dateien im Kontext unseres Industriepartners bewährt, da dessen zentrales EA-Repository alle verfügbaren Daten über TAPs als CSV-Export zur Verfügung stellt. Demnach haben wir für unseren Prototypen einen CSV-Importer realisiert, wobei auch Importer für andere Datenquellen (z.B. für Datenbanken oder Webservices) implementiert werden können (vgl. Unterabschnitt 3.1.2). Als Output erzeugt unser Softwareprototyp einen konkreten *Restrukturierungsplan*, welcher alle Informationen zu den erforderlichen Migrations- und Anpassungsmaßnahmen für die analysierten TAPs beinhaltet und für Experten anschaulich visualisiert.

Im folgenden werden die einzelnen Oberkategorien näher erläutert, indem eine Beschreibung ihrer Funktionalität, ihrer benötigten Schnittstellen sowie der zur Realisierung eingesetzten Technologien gegeben wird. Ist dabei für solch eine Komponente keine weitere Technologie angegeben, so wurde ihre Funktionalität ohne zusätzliche Bibliotheken oder Plugins mit reinen Java-Mitteln implementiert.

Variability Analysis	
Funktion:	Realisierung der im Kapitel 3 vorgestellten Ansätze zur Datenaufbereitung und Variabilitätsbestimmung für die importierten TAPs. Stellt hierzu geeignete CSV-Importer für die Datentransformation von `TAPs` und den dazugehörigen `Categories` bereit
Technologie:	*Eclipse Modeling Framework (EMF)* zur Modellierung von Ecore-Metamodellen und *Family Mining Framework* [Wil18] zur Realisierung der Variabilitätsbestimmung
Benötigte Schnittstellen:	keine, benötigt aber die beiden zu importierenden CSV-Dateien `TAPs` und `Categories`

Recommendation Deriviation	
Funktion:	Realisierung der im Kapitel 4 vorgestellten Ansätze für die regelbasierte Businessanalyse, die graphbasierte Technologieanalyse sowie der Ableitung konkreter Restrukturierungsempfehlungen
Technologie:	Java
Benötigte Schnittstellen:	Benötigt das generierte 150%-Modell für die analysierten TAPs aus der *Variability Analysis* sowie die spezifizierten Geschäftsregeln und eine eventuell definierte Graphbeschreibung aus der Oberkomponente *DSL Provider*

Recommendation Assessment	
Funktion:	Umsetzung der im Kapitel 5 präsentierten Methoden für die automatisierte Einzelbewertung von Restrukturierungsempfehlungen und die automatisierte Portfoliobewertung für das gesamte Set aller Restrukturierungsempfehlungen
Technologie:	*Delta Ecore* [SSA14] zur Generierung von spezifischen Deltaoperationen für die einzelnen Restrukturierungsempfehlungen
Benötigte Schnittstellen:	Benötigt die entworfene Deltasprache für die Generierung von konkreten Deltaoperationen, die spezifizierte Aufwandsmatrix und das spezifizierte Abhängigkeitsmodell aus der Oberkomponente *DSL Provider*. Erfordert zudem das generierte 150%-Modell für die Differenzanalyse aus der *Variability Analysis* sowie das Set aller abgeleiteten Restrukturierungsempfehlungen aus der *Recommendation Deriviation*

DSL Provider	
Funktion:	Realisierung der entworfenen Domain-Specific Languages (DSLs) zur Spezifizierung von Geschäftsregeln, Graphbeschreibungen, Aufwandsmatrizen und Abhängigkeitsmodellen. Außerdem erfolgt hier die Umsetzung der entworfenen Delta Sprache zur Spezifizierung von Deltaoperationen für die Transformation eines 150%-Modells
Technologie:	*Xtext* zur Entwicklung von Domänenspezifischen Sprachen sowie *Delta Ecore* zur Spezifizierung der Delta Sprache und ihrem Set an erforderlichen Deltaoperationen
Benötigte Schnittstellen:	keine

Simulation & Restructuring	
Funktion:	Realisierung der Ansätze für die Simulation von ausgewählten Restrukturierungsempfehlungen (vgl. Unterabschnitt 5.3.2) und die Restrukturierungsplanung (vgl. Abschnitt 5.4) zur Erzeugung eines konkreten Restrukturierungsplans im HTML-Ausgabeformat
Technologie:	*Delta Ecore* zur automatisierten Transformation eines konkreten 150%-Modells und *HTML* zur Erstellung eines strukturierten Dokumentes für die spätere Darstellung des Restrukturierungsplans
Benötigte Schnittstellen:	Erfordert die zur Simulation ausgewählten Restrukturierungsempfehlungen aus der Oberkomponente *Recommendation Assessment* und das generierte 150%-Modell für die Transformation aus der *Variability Analysis*

Viewer	
Funktion:	Kapselt die grafischen Oberflächen zur Darstellung der erzeugten Informationen und zur Interaktion mit dem Nutzer
Technologie:	*Graphical Editing Framework (GEF)* zur Erstellung von grafischen Oberflächen für Java-Applikationen und *Java Universal Network/Graph Framework (JUNG)* zur Visualisierung von Graphen
Benötigte Schnittstellen:	Benötigt die Ergebnisse aus allen anderen Oberkomponenten zur visuellen Darstellung und Interaktion mit dem Nutzer

Die in Abbildung 6.1 gezeigte Architektur unseres Softwareprototypen ist modular aufgebaut und bietet dadurch einige Vorteile. Diese werden nachfolgend kurz mit Blick auf die formulierten Architekturziele erläutert.

Durch den anwendungsbezogenen Fokus beziehungsweise die *Kapselung* von Funktionalität in den einzelnen Komponenten entsteht ein übersichtliches Bild der Gesamtarchitektur, sodass der Zugriff auf den Source-Code bestimmter Funktionen schnell und gezielt erfolgen kann. So lässt sich beispielsweise die Methode `calculateSimilarity(graph1, graph2)` zur Bestimmung der Ähnlichkeit zweier VKGs in der Komponente *Technology Analysis* finden, welche der Oberkomponente *Recommendation Deriviation* zugeordnet ist. Hierdurch kann eine *effiziente Wartung und Weiterentwicklung* aller Bestandteile des entworfenen Softwareprototypen erfolgen. Zudem werden in der Oberkomponente *DSL Provider* alle von Experten zu spezifizierenden Artefakte (z.B. Geschäftsregeln) gebündelt. Dies ermöglicht eine *gezielte Anpassung* des vorgestellten Ansatzes an unternehmensindividuelle Rahmenbedingungen innerhalb einer gemeinsamen Oberkomponente.

Darüber hinaus wird auch die *Wiederverwendung* von Source-Code zur Entwicklungszeit und von Artefakten zur Laufzeit unterstützt. Zum Beispiel findet das entworfene Metamodell für die TAPs, welches sich in der spezifischen Komponente *Data Model* (vgl. Abschnitt A.3) befindet, überall Anwendung. So kann die *Variability Analysis* ein generiertes 150%-Modell nach außen anbieten und alle anderen Komponenten können dieses für ihre weitere Verarbeitung nutzen (z.B. für die Differenzanalyse in der Komponente *Recommendation Assessment*).

Ein weiterer Vorteil dieser Architektur ist die einfache *Erweiterbarkeit* des Softwareprototypen. Hierdurch können neu entworfene Funktionen mit Hilfe einer zusätzlichen Komponente sehr einfach in die bestehende Architektur integriert werden. So wäre es beispielsweise möglich, eine weitere Komponente zur Identifizierung von Architekturmustern in den betrachteten TAPs zu entwickeln und durch geeignete Schnittstellen zur *Variability Analysis* und zum *Viewer* in die vorhandene Architektur zu integrieren. So würde die neue Komponente mit den nötigen Datenmodellen für die eigene Analyse versorgt werden und könnte ihre Ergebnisse zur Darstellung und Interaktion mit dem Nutzer ebenfalls an den *Viewer* übergeben.

Detailbeschreibung der Oberkomponente Viewer

Die vorgestellte Oberkomponente *Viewer* dient dazu, alle Ergebnisse aus den anderen Komponenten der gezeigten Architektur unseres Softwareprototypen in einer für Domänenexperten geeigneten Art und Weise darzustellen und ihnen darüber die Interaktion mit dem System zu ermöglichen. Zu diesem Zweck haben wir eine graphische Oberfläche entwickelte, welche alle erzeugten Informationen in spezifischen Views darstellt und in eine Gesamtsicht integriert. Die Abbildung 6.3 zeigt diese graphische Oberfläche, welche im nachfolgenden kurz erläutert wird.

Bereich A: Stellt jeweils einen Reiter für jedes 150%-Modell bereit. Hierbei kann es sich entweder um verschiedene Ist-150%-Modelle unterschiedlicher Sets von analysierten TAPs oder um aufeinander aufbauende Soll-150%-Modelle des gleichen Sets von betrachteten TAPs handeln. Über die Reiter kann leicht zwischen den einzelnen 150%-Modellen gewechselt werden.

Bereich B: Stellt das jeweils ausgewählte 150%-Modell grafisch dar. Dabei werden unterschiedliche Detailinformationen veranschaulicht, insbesondere die verbauten Komponenten, ihre Variabilitätsinformationen sowie Nutzungshäufigkeiten. Darüber hinaus werden auch potentielle Ausreißer, welche über die Ausreißersuche identifiziert worden sind, durch einen gelben Kreis markiert. Durch einen Klick auf diesen Kreis werden alternative Tiers als Vorschlag in einer schwarzen Box angezeigt. Wählt ein Nutzer einen dieser Vorschläge aus, so wird das entsprechende Element verschoben und das 150%-Modell neu generiert. Aus Gründen der Übersichtlichkeit werden im Bereich B nur logische Elemente angezeigt. Durch einen Klick auf eines dieser Elemente öffnet sich aber die zusätzliche Sicht *DetailView* (Abbildung 6.2), welche alle physischen Elemente und ihre Verbreitung in den jeweiligen TAPs für das ausgewählte logische Element anzeigt.

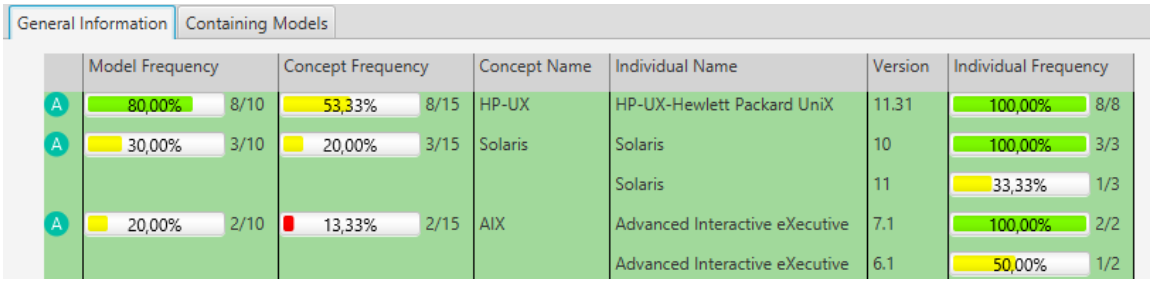


Abbildung 6.2: Screenshot der grafischen Oberfläche *DetailView*

Bereich C: Bietet unterschiedliche Steuerungselemente, um beispielsweise die Generierung von Restrukturierungsempfehlungen für das aktuell im Bereich B dargestellte 150%-Modell anzustoßen oder die Simulation der im Bereich D ausgewählten Restrukturierungsempfehlungen durchzuführen.

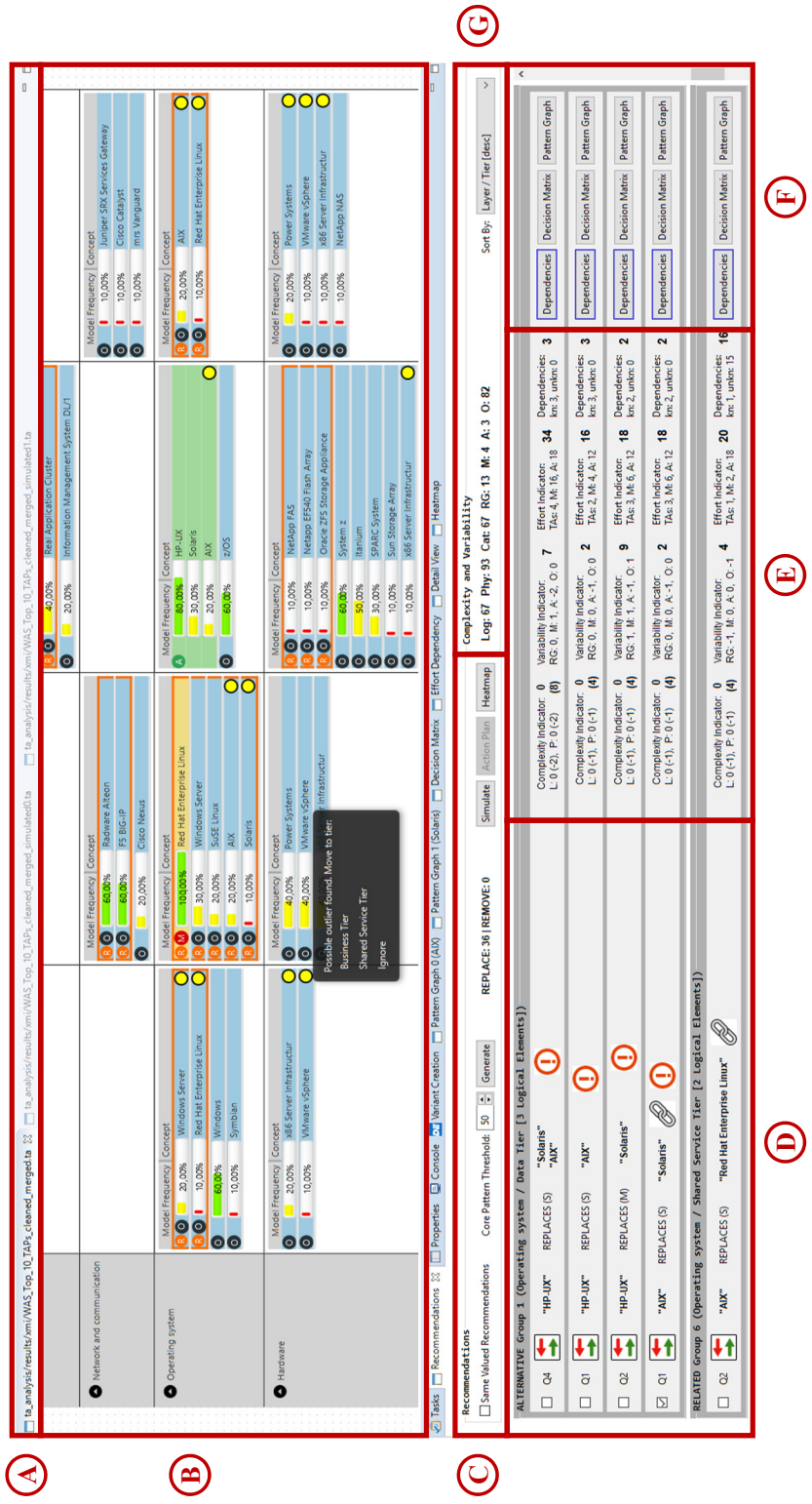
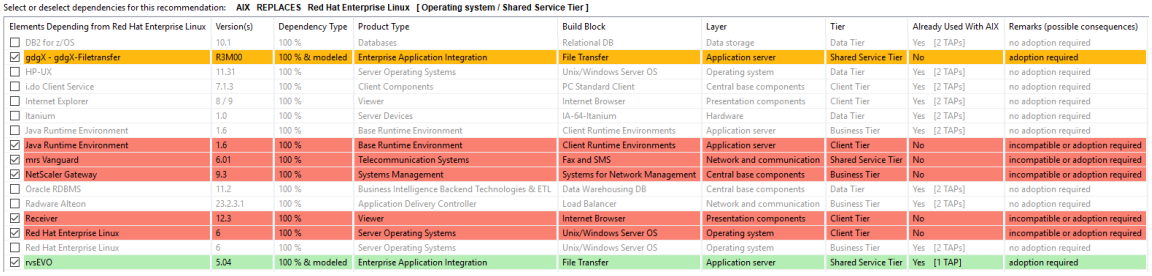


Abbildung 6.3: Screenshot der grafischen Oberfläche der Komponente *Viewer*

Bereich D: Listet alle abgeleiteten Ersetzungs- und Entfernungsempfehlungen auf. Dabei werden für jede Empfehlung unter anderem auch die Zuordnung zum jeweiligen Quadranten der Aufwand-Nutzen-Matrix sowie ihre Korrelationen zu anderen Restrukturierungsempfehlungen gezeigt.

Bereich E: Stellt Informationen über die Ergebnisse aus der Einzelbewertung für jede Restrukturierungsempfehlung dar. Diese beinhalten beispielsweise die Variabilitätsdifferenz, die ermittelten Migrations- und Anpassungsaufwände sowie die Anzahl der Abhängigkeiten.

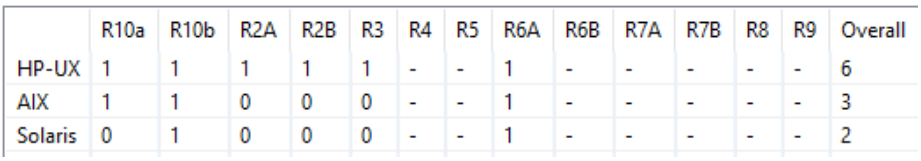
Bereich F: Bietet weitere Steuerungselemente, mit denen auf spezifische Detailsichten für jede einzelne Restrukturierungsempfehlung gewechselt werden kann. So lassen sich beispielsweise in der *DependencyView* (Abbildung 6.4) alle abhängigen Elemente einer konkreten Restrukturierungsempfehlung betrachten und durch einen Domänenexperten verifizieren.


Select or deselect dependencies for this recommendation: AIX REPLACES Red Hat Enterprise Linux [Operating system / Shared Service Tier]

Elements Depending from Red Hat Enterprise Linux	Version(s)	Dependency Type	Product Type	Build Block	Layer	Tier	Already Used With AIX	Remarks (possible consequences)
☐ DB2 for z/OS	10.1	100 %	Databases	Relational DB	Data storage	Data Tier	Yes [2 TAPs]	no adoption required
☑ gdgX - gdgX-Filetransfer	R3M00	100 % & modeled	Enterprise Application Integration	File Transfer	Application server	Shared Service Tier	No	adoption required
☐ HP-UX	11.31	100 %	Server Operating Systems	Unix/Windows Server OS	Operating system	Data Tier	Yes [2 TAPs]	no adoption required
☐ Ido Client Service	7.1.3	100 %	Client Components	PC Standard Client	Central base components	Client Tier	Yes [2 TAPs]	no adoption required
☐ Internet Explorer	8 / 9	100 %	Viewer	Internet Browser	Presentation components	Client Tier	Yes [2 TAPs]	no adoption required
☐ Itanium	1.0	100 %	Server Devices	IA-64-Itanium	Hardware	Data Tier	Yes [2 TAPs]	no adoption required
☐ Java Runtime Environment	1.6	100 %	Base Runtime Environment	Client Runtime Environments	Application server	Business Tier	Yes [2 TAPs]	no adoption required
☑ Java Runtime Environment	1.6	100 %	Base Runtime Environment	Client Runtime Environments	Application server	Client Tier	No	incompatible or adoption required
☑ mrs Vanguard	6.01	100 %	Telecommunication Systems	Fax and SMS	Network and communication	Shared Service Tier	No	incompatible or adoption required
☑ NetScaler Gateway	9.3	100 %	Systems Management	Systems for Network Management	Central base components	Business Tier	No	incompatible or adoption required
☐ Oracle RDBMS	11.2	100 %	Business Intelligence Backend Technologies & ETL	Data Warehousing DB	Central base components	Data Tier	Yes [2 TAPs]	no adoption required
☐ Radware Alteon	23.2.3.1	100 %	Application Delivery Controller	Load Balancer	Network and communication	Business Tier	Yes [2 TAPs]	no adoption required
☑ Receiver	12.3	100 %	Viewer	Internet Browser	Presentation components	Client Tier	No	incompatible or adoption required
☑ Red Hat Enterprise Linux	6	100 %	Server Operating Systems	Unix/Windows Server OS	Operating system	Client Tier	No	incompatible or adoption required
☐ Red Hat Enterprise Linux	6	100 %	Server Operating Systems	Unix/Windows Server OS	Operating system	Business Tier	Yes [2 TAPs]	no adoption required
☑ rvsEVO	5.04	100 % & modeled	Enterprise Application Integration	File Transfer	Application server	Shared Service Tier	Yes [1 TAP]	adoption required

Abbildung 6.4: Screenshot der grafischen Oberfläche *DependencyView*

Darüber hinaus ist es durch die *DecisionMatrixView* (vgl. Abbildung 6.5) auch möglich, die Entscheidungsmatrix aus der Businessanalyse einzusehen. Zudem können die jeweiligen VKGs einer betrachteten Restrukturierungsempfehlung mit Hilfe der *GraphView* (vgl. Abbildung 6.6) inspiziert werden.



	R10a	R10b	R2A	R2B	R3	R4	R5	R6A	R6B	R7A	R7B	R8	R9	Overall
HP-UX	1	1	1	1	1	-	-	1	-	-	-	-	-	6
AIX	1	1	0	0	0	-	-	1	-	-	-	-	-	3
Solaris	0	1	0	0	0	-	-	1	-	-	-	-	-	2

Abbildung 6.5: Screenshot der grafischen Oberfläche *DecisionMatrixView*

Bereich G: Hier werden die Werte der aktuell gemessenen Variabilität für das im Bereich B betrachtete 150%-Modell dargestellt. Zudem wird auch eine Auswahlbox zur Sortierung der im Bereich D angezeigten Liste von Restrukturierungsempfehlungen angeboten. Eine Sortierung kann dabei aufsteigend oder absteigend nach der Zuordnung zum *Layer/Tier* oder nach einem der Werte aus der Einzelbewertung, welche im Bereich E angezeigt werden, erfolgen.

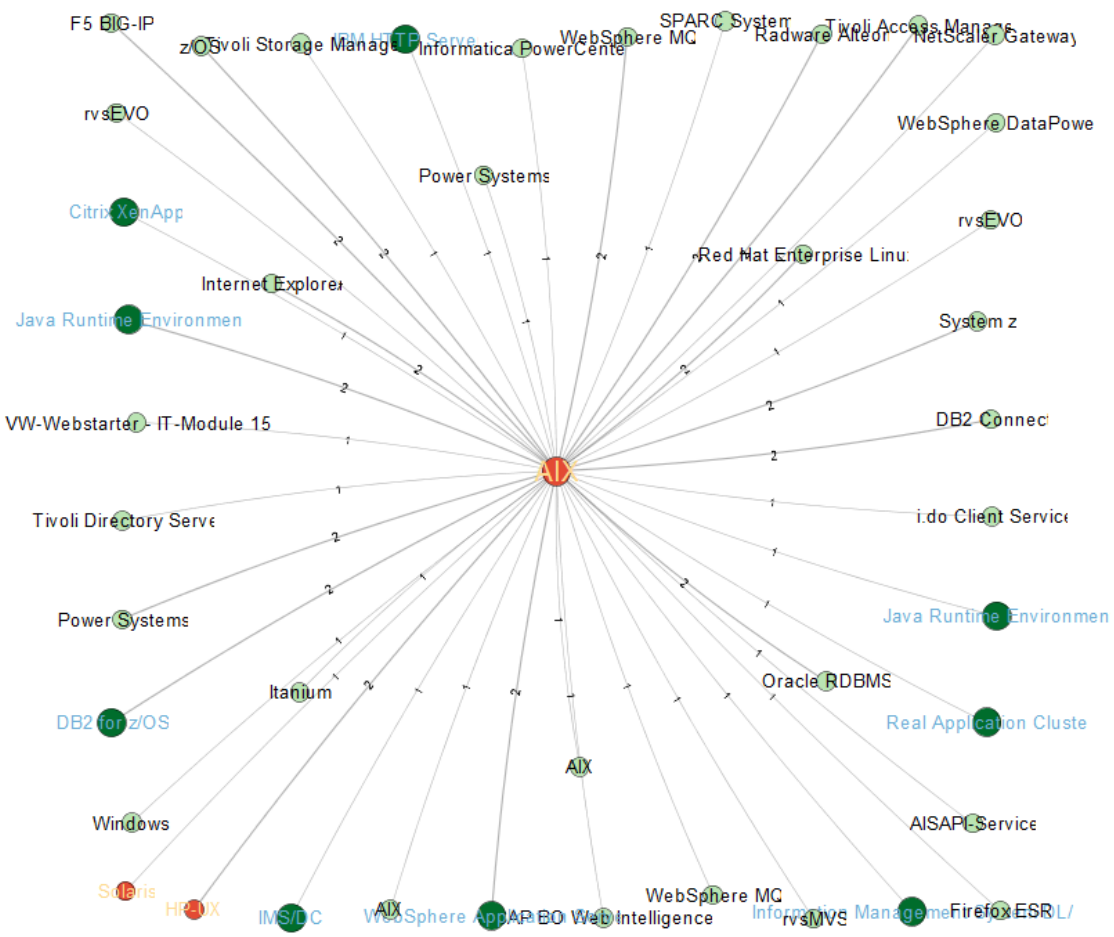


Abbildung 6.6: Screenshot der grafischen Oberfläche *GraphView*

6.2 Fallstudien

Zur Evaluation unseres Ansatzes wurden drei aufeinander aufbauende Fallstudien durchgeführt, welche in diesem Abschnitt vorgestellt werden. Für alle drei Fallstudien ist dabei dieselbe Datenbasis verwendet worden. Diese wird zunächst im Unterabschnitt 6.2.1 erläutert. Anschließend werden die einzelnen Fallstudien A, B und C in den Abschnitten 6.2.2 bis 6.2.4 beschrieben und ihre Ergebnisse diskutiert.

6.2.1 Daten für die Fallstudien

Für unsere Fallstudien hatten wir Zugriff auf das zentrale EA-Repository unseres Industriepartners, das unternehmensweit alle wichtigen Informationen, wie zum Beispiel die dokumentierten TAPs der Softwaresysteme aller Geschäftsbereiche und Standorte enthält. Da diese Informationen vertraulich sind, können wir in diesem Abschnitt nur aggregierte und abstrahierte Details über die analysierten Daten präsentieren.

	Set A	Set B	Set C	Set D
Kerntechnologie	WebSphere Application Server	Internet Information Server	Oracle Datenbank	IBM DB2
Anzahl TAPs	369	429	442	262

Tabelle 6.1: Analysierte TAPs im Rahmen der Fallstudien

Zusammen mit zwei Architekten[3] unseres Industriepartners haben wir die verfügbaren Daten manuell analysiert und auf Grundlage der Vorgehensweise in Unterabschnitt 3.1.1 eine Auswahl an TAPs für die Verarbeitung im Rahmen der Fallstudien getroffen. Um dabei die Aussagekraft der Fallstudienergebnisse zu erhöhen, sollte jede Fallstudie mehrfach mit unterschiedlichen TAPs durchgeführt werden. Aus diesem Grund haben wir vier verschiedene Kerntechnologien als *Verwendungszweck* (vgl. Unterabschnitt 3.1.1) definiert und auf dieser Basis vier unabhängige Sets von verwandten TAPs gebildet, welche die folgende Tabelle 6.1 präsentiert.

Wie in Tabelle 6.1 zu erkennen ist, konzentrieren sich die beiden *Sets A* und *B* auf TAPs, die zwei verschiedene *Web-Server* implementiert haben. So konnten dem Set A insgesamt 369 TAPs mit einem `WebSphere Application Server` und dem Set B zusammen 429 TAPs mit einem `Internet Information Server` zugeordnet werden. Die beiden *Sets C* und *D* berücksichtigen hingegen TAPs mit zwei unterschiedlichen *Datenbanksystemen*. Für das Set C konnten so 442 TAPs mit einer `Oracle Datenbank` und für das Set D insgesamt 262 TAPs mit einer `IBM DB2` Datenbank identifiziert werden.

Um eine bessere Vergleichbarkeit der Fallstudienergebnisse für die einzelnen Sets zu ermöglichen, haben wir die TAPs in den jeweiligen Sets anhand ihrer Größe in absteigender Reihenfolge sortiert. Dabei wurde ihre Größe durch die Anzahl der enthaltenen Komponenten bestimmt. Anschließend konnten die sortierten TAPs in jedem einzelnen Set in Teilmengen von *Top10* bis *Top320* zusammengefasst werden. So enthält beispielsweise die Menge *Top120* aus dem Set `IBM DB2` die 120 größten TAPs dieses Sets. Für das Set D konnten die Teilmengen *Top280* und *Top320* nicht gebildet werden, da dieses Set lediglich 262 TAPs beinhaltet. Die folgende Tabelle 6.2 zeigt eine Übersicht über die gebildeten Teilmengen mit der Anzahl der enthaltenen Komponenten pro Teilmenge sowie pro TAP in der jeweiligen Teilmenge.

In der Tabelle 6.2 werden die aggregierten Informationen zu den gebildeten Teilmengen für die vier Sets dargestellt. Zu jeder Teilmenge sind ihre minimale, maximale und durchschnittliche Anzahl an Komponenten sowie die durchschnittliche Anzahl der Elemente pro TAP angegeben. So sind beispielsweise in allen vier *Top10-*

[3] für die Durchführung der Fallstudien und der Experteninterviews standen dieselben 6 Architekten zur Verfügung (vgl. Unterabschnitt 6.3.1)

			Anzahl der Komponenten pro Teilmenge			pro TAP
Teilmenge	**N**	**Sets**	**min**	**max**	**∅**	**∅**
Top 10	4	A–D	198	289	258,8	25,9
Top 20	4	A–D	375	506	464,3	23,2
Top 40	4	A–D	645	862	800,8	20,0
Top 80	4	A–D	1.127	1.462	1.370,0	17,1
Top 120	4	A–D	1.578	1.946	1.839,8	15,3
Top 160	4	A–D	1.962	2.336	2.220,3	13,9
Top 200	4	A–D	2.301	2.659	2.536,5	12,7
Top 240	4	A–D	2.583	2.981	2.789,3	11,6
Top 280	3	A–C	2.826	3.243	3.049,0	10,9
Top 320	3	A–C	3.029	3.445	3.233,7	10,1

N = Anzahl

Tabelle 6.2: Gebildete Teilmengen in den einzelnen Sets A–D

Teilmengen der *Sets A–D* im Durchschnitt 258,8 Komponenten beinhaltet, wobei die kleinste Top10-Teilmenge 198 und die größte 289 Elemente umfasst. Die einzelnen TAPs in dieser Teilmenge haben im Durchschnitt 25,9 Komponenten verbaut.

Die Fallstudien wurden mit Hilfe des entwickelten Softwareprototypen auf Basis der hier vorgestellten Daten durchgeführt. Welche der hier präsentierten Teilmengen dabei in den jeweiligen Fallstudien verwendet wurden, ist in den nachfolgenden Abschnitten zu den Fallstudien näher beschrieben.

6.2.2 Fallstudie A: Identifizierung von Variabilität

Im Rahmen der Fallstudie A wurde unser automatisierter Ansatz zur Identifizierung der Variabilität von betrachteten TAPs (vgl. Kapitel 3) evaluiert. Hierbei standen die folgenden Forschungsfragen (FF) im Fokus:

FF-A.1 – Übereinstimmung mit den Erwartungen: *Entsprechen die identifizierten Variabilitätsinformationen den Ergebnissen, die von den mit der Domäne vertrauten Architekten erwartet werden?*

FF-A.2 – Laufzeit: *Ist die Ausführungszeit für den Variabilitätsmining-Algorithmus in einem akzeptablen Zeitbereich?*

FF-A.3 – Skalierbarkeit: *Skaliert der Variabilitätsmining-Algorithmus für Szenarien mit einer wachsenden Anzahl von TAPs?*

Für die Evaluierung der Ergebnisse des Variabilitätsmining-Ansatzes standen zwei Domänenexperten unseres Industriepartners zur Verfügung. Das vorgeschlagene

Cross-Sets	N	vereinigte Sets	Anzahl der Komponenten pro Teilmenge min	max	∅	pro TAP ∅
Cr. 10	2	A∩B, C∩D	472	563	517,5	25,9
Cr. 20	2	A∩B, C∩D	862	995	928,5	23,2
Cr. 640	1	A∩B	6.474		6.474,0	10,1
Cr. 960	1	A∩B∩C	9.701		9.701,0	10,1
Cr. 1200	1	A∩B∩C∩D	12.405		12.405,0	10,3

N = Anzahl

Tabelle 6.3: Zusätzlich gebildete Cross-Sets für die Fallstudie A

Mining-Verfahren wurde mit Hilfe des entwickelten Softwareprototypen durchgeführt. Mit diesem wurden alle zur Verfügung stehenden Daten, welche in der Tabelle 6.2 dargestellt sind, verarbeitet. Um aussagekräftigere Evaluationsergebnisse in Bezug auf die Skalierbarkeit und die Laufzeit unseres Variabilitätsmining-Ansatzes zu erhalten, wurden weitere Teilmengen aus den verfügbaren Daten generiert. So sind zusätzlich zu den *Top*-Mengen in Tabelle 6.2 die in Tabelle 6.3 gezeigten *Cross-Sets Cr.10* bis *Cr.1200* gebildet worden, welche gleiche Top-Teilmengen der unterschiedlichen Sets A–D miteinander kombinieren.

Wie in Tabelle 6.3 zu sehen ist, wurden zwei Cross-Sets *Cr.10*, zwei weitere Cross-Sets *Cr.20* sowie jeweils ein Cross-Set *Cr.640*, *Cr.960* und *Cr.1200* gebildet. Die Cross-Sets *Cr.10* und *Cr.20* sind jeweils durch Vereinigung der *Top10*- und *Top20*-Mengen der Sets A und B sowie C und D entstanden, sodass hierdurch Kerntechnologien mit ähnlichem Fokus kombiniert wurden (*Webserver* beziehungsweise *Datenbanksysteme*). Für das Cross-Set *Cr.640* sind die beiden *Top320*-Mengen aus den Sets A und B vereinigt worden. Diese sind ebenfalls im Cross-Set *Cr.960* enthalten, welches zudem die *Top320*-Teilmenge aus dem Set C umfasst. Für das Cross-Set *Cr.1200* wurde den Teilmengen in *Cr.960* zudem noch die *Top240*-Menge aus dem Set D hinzugefügt.

Insgesamt stehen damit 45 Sets von ausgewählten TAPs für die Evaluierung unseren Variabilitätsmining-Verfahrens zur Verfügung. Dabei beinhaltet das größte Set (*Cr.1200*) insgesamt 1.200 unterschiedliche TAPs mit mehr als 12.400 Komponenten.

Methodik

Die automatische Analyse der 45 Sets mit den ausgewählten TAPs erlaubt eine effektive Bewertung der Laufzeit und Skalierbarkeit unseres Ansatzes für große, gewachsene Technologiearchitekturen. Daher haben wir unseren Variabilitätsmining-Algorithmus mit Hilfe des Softwareprototypen für jedes dieser 45 Sets ausgeführt und dabei die jeweilige Ausführungszeit gemessen. Um dabei den Einfluss einer ungenauen Laufzeitmessung zu reduzieren, wurde das Variabilitätsmining wiederholt

und für jedes Set insgesamt 10 Mal durchgeführt. Zur Ausführung wurde ein Laptop mit 2,7 GHz Intel i7 und 12 GB RAM verwendet.

Darüber hinaus haben wir eine manuelle Analyse von ausgewählten Sets mit Unterstützung von Domänenexperten durchgeführt, um zu bewerten, ob die generierten Ergebnisse der Variabilitätsanalyse den Erwartungen der Experten entsprechen und somit im industriellen Kontext von Restrukturierungen anwendbar sind. Da eine vollständige manuelle Analyse jedes TAPs und aller Ergebnisse aus der Variabilitätsanalyse aufgrund der hohen Anzahl an Komponenten nicht möglich war, haben sich die Domänenexperten auf die Analyse der 12 *Top10*-, *Top20*-, *Cr.10*- und *Cr.20*-Mengen konzentriert. Zusätzlich wurden auch größere Sets stichprobenartig manuell überprüft, um unseren Ansatz zu evaluieren. Im Rahmen dieser manuellen Analyse haben die Domänenexperten zuerst die einzelnen TAPs eines ausgewählten Sets gesichtet und anschließend das entsprechende Ergebnis der Variabilitätsanalyse für das jeweils betrachtete Set bewertet. Hierfür haben wir den Experten die resultierenden 150%-Modelle vorgestellt und die enthaltenen Informationen erläutert. Um diese zu bewerten, wurden die folgenden Faktoren durch die Experten berücksichtigt:

1. *Vollständigkeit*: Das generierte 150%-Modell enthält alle Komponenten aus den eingegebenen TAPs des betrachteten Sets.
2. *Korrektheit*: Die manuell identifizierten Variabilitätsbeziehungen stimmen mit den automatisch generierten Ergebnissen unseres Algorithmus überein.

Ergebnisse und Diskussion

Im Folgenden stellen wir die Ergebnisse unserer Fallstudie vor und diskutieren diese im Hinblick auf unsere Forschungsfragen.

FF-A.1 – Übereinstimmung mit den Erwartungen: Während der Überprüfung der Ergebnisse aus der automatisierten Analyse identifizierten die Domänenexperten zunächst einige Variabilitätsbeziehungen, die ihren Erwartungen widersprachen. Hierbei handelte es sich um die in Unterabschnitt 3.2.2 beschriebenen Szenarien. Nach Anwendung der vorgestellten Splitting-Regeln bestätigten sie jedoch, dass die generierte Variabilitätsinformationen für alle Sets vollständig und korrekt sind. So enthalten die erzeugten 150%-Modelle alle Komponenten aus den eingegeben TAPs und stellen Variabilitätsbeziehungen dar, die mit den Ergebnissen aus der manuellen Analyse der Domänenexperten übereinstimmen. Somit können wir *FF-A.1* positiv beantworten.

FF-A.2 – Laufzeit: Während der Evaluierung haben wir die Laufzeit unseres Variabilitätsmining-Ansatzes für die ausgeführten Sets gemessen. In Abbildung 6.7 präsentieren wir die Ergebnisse dieser Messung als Boxplots für die jeweiligen *Top10*- bis *Top320*-Mengen. Dabei enthalten die Boxplots für die Mengen mit 20 und 40 TAPs auch die beiden Cross-Sets *Cr.10* (mit 20 TAPs) und *Cr.20* (mit 40 TAPs).

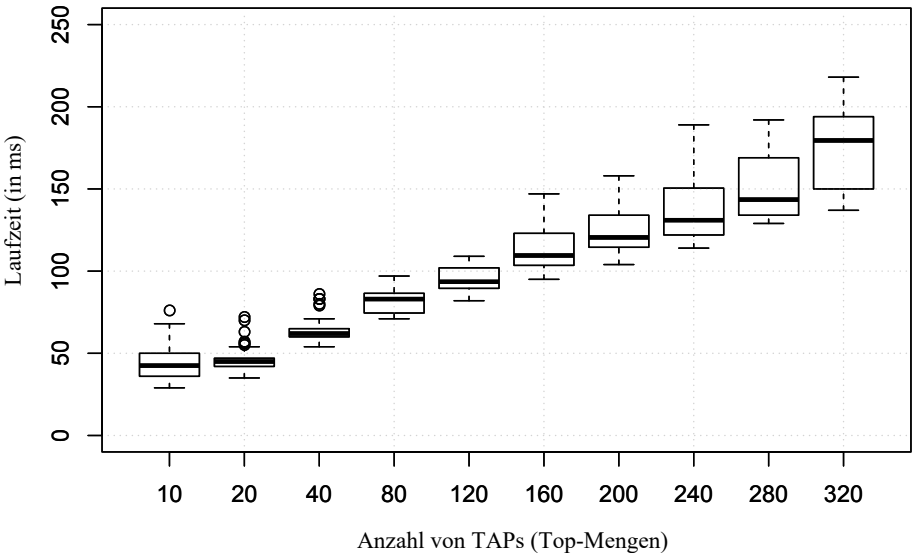


Abbildung 6.7: Boxplots zur Darstellung der Laufzeit für Top 10 – Top 320-Mengen

Jeder dieser Boxplots zeigt die aggregierten Laufzeiten für die Analyse-Schritte *Clustering*, *Splitting* und *Merging* unseres Ansatzes (vgl. Abschnitt 3.2). Die Laufzeiten für das Importieren der TAPs aus dem Format unseres Industriepartners in unser internes Datenformat und das Exportieren des 150%-Modells als Ergebnis der Analyse werden in den Boxplots nicht berücksichtigt, da wir uns auf die Leistung des Mining-Algorithmus konzentrieren wollten. Wir haben diese Laufzeiten jedoch für das größte Sets *Cr.1200* gemessen, um die maximalen Import- und Export-Laufzeiten zu bestimmen. Im Ergebnis haben wir für das Cross-Set *Cr.1200* eine Importzeit von durchschnittlich 4.336 ms und eine Exportzeit von durchschnittlich 82 ms festgestellt. Bei Betrachtung der Boxplots lässt sich feststellen, dass der Variabilitätsmining-Algorithmus Laufzeiten von etwa 30 ms bis etwa 220 ms für die ausgeführten Mengen mit 10 bis 320 TAPs aufweist. Zudem haben wir eine maximale Laufzeit für das Cross-Sets *Cr.640* von 450 ms, für das Cross-Set *Cr.960* von 568 ms und für das größte Cross-Set *Cr.1200* von 592 ms gemessen (vgl. Abbildung 6.8).

Aus den Messergebnissen lässt sich schließen, dass die Laufzeiten für unseren Variabilitätsmining-Algorithmus innerhalb eines akzeptablen Bereiches für solche industriell gewachsenen Technologiearchitekturen liegen, selbst wenn die maximalen Import- und Exportzeiten berücksichtigt werden. Die an der Fallstudie beteiligten Experten haben diese Einschätzung bestätigt. Einer von ihnen hat sogar betont, dass auch viel längere Laufzeiten akzeptabel wären, da Technologiearchitekturen und auch TAPs einzelner Systeme nicht täglich geändert werden. Daher wäre es möglich, den Variabilitätsmining-Algorithmus nach größeren Änderungen erneut auszuführen und diesen über Nacht auf leistungsfähigeren Servern des Industriepartners laufen zu lassen. Die beobachteten niedrigen Laufzeiten können durch den geringen Speicherbedarf der analysierten Datenstrukturen erklärt werden (vgl. Unterabschnitt 3.1.2), was

darauf zurückzuführen ist, dass unser Algorithmus nur begrenzte Details benötigt. Darüber hinaus reduziert die frühzeitige Eliminierung unrealistischer Variabilitätsbeziehungen die Anzahl der durchzuführenden Vergleiche, was ebenfalls zu kürzen Laufzeiten führt (vgl. Unterabschnitt 3.2.1). Zusammenfassend können wir demnach auch die Forschungsfrage *FF-A.2* bestätigen.

FF-A.3 – Skalierbarkeit: Um die Skalierbarkeit unseres Ansatzes zu bewerten, setzen wir die gemessenen Laufzeiten in Relation zur Anzahl der verglichenen TAPs. Genau wie bei der Laufzeitanalyse betrachteten wir auch hierbei aggregierte Laufzeiten für die drei Analyse-Schritte *Clustering*, *Splitting* und *Merging*, die in Abschnitt 3.2 beschrieben sind. In Abbildung 6.8 ist das resultierende Streudiagramm dargestellt. Dieses beinhaltet zusätzlich eine Trendlinie, die den Anstieg der Laufzeiten mit einer wachsenden Anzahl von verglichenen TAPs visualisiert.

Wie in Abbildung 6.8 zu erkennen ist, sind alle Datenpunkte mehr oder weniger gleichmäßig um diese Trendlinie in einem Intervall von ungefähr ±100 ms verstreut. Lediglich für Sets über 320 TAPs sind größere Streuungen von bis zu ±150 ms zu erkennen. Insgesamt lässt sich jedoch feststellen, dass die erzeugte Trendlinie einem linearen Trend folgt. Allerdings befindet sich der Großteil aller gemessenen Laufzeiten in dem Bereich der *Top10*- bis *Top320*-Mengen. Umfangreichere Sets konnten nur in drei Fällen mit 640, 960 bzw. 1200 TAPs bewertet werden. Daher kann die Signifikanz der Datenpunkte für die größeren Sets mit über 320 TAPs begrenzt sein. Dennoch sind wir überzeugt, dass die analysierten Sets eine ausreichend große Menge an Komponenten berücksichtigen. So beinhalten die betrachteten Sets bis zu 12.405 Komponenten und umfassen TAPs mit einer durchschnittliche Anzahl von etwa 10 bis 26 Komponenten (vgl. Tabelle 6.2). Unter Berücksichtigung dieser Tatsache können

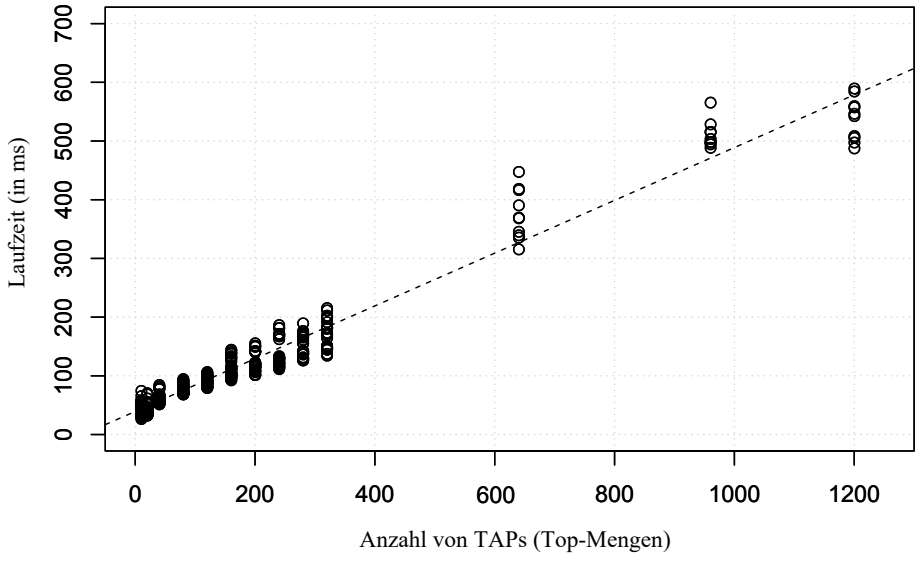


Abbildung 6.8: Streudiagramm zur Darstellung der Skalierbarkeit unseres Variabilitätsmining-Algorithmus

wir argumentieren, dass die betrachteten Mengen repräsentativ für große, gewachsene Technologiearchitekturen sind und realistische Fälle widerspiegeln. Daher können wir auch *FF-A.3* positiv beantworten.

6.2.3 Fallstudie B: Ableitung von Restrukturierungsempfehlungen

Im Rahmen der Fallstudie B stand die Evaluierung unseres automatisierten Ansatzes für die Ableitung von Restrukturierungsempfehlungen zur Reduzierung unnötiger Variabilität in Sets von ausgewählten TAPs (vgl. Kapitel 4) im Fokus. Hierbei waren die folgenden Forschungsfragen (FF) relevant:

FF-B.1 – Allgemeiner Ansatz: *Ist der vorgeschlagene Ansatz geeignet, Restrukturierungsempfehlungen für unterschiedliche Sets von ausgewählten TAPs abzuleiten?*

FF-B.2 – Restrukturierungsempfehlungen: *Sind die abgeleiteten Restrukturierungsempfehlungen geeignet, um Experten bei der Reduzierung unnötiger Variabilität zu unterstützen?*

Zur Durchführung dieser Fallstudie standen drei Domänenexperten unseres Industriepartners zur Verfügung. Darüber hinaus haben wir die jeweiligen *Top20*-Mengen der *Sets A–D* verwendet und diese mit Hilfe unseres Softwareprototypen verarbeitet. Für jedes Set wurde hierfür zunächst die Variabilitätsanalyse durchgeführt, um das erforderliche 150%-Modell als Basis für die Ableitung von Restrukturierungsempfehlungen zu generieren. Die nachfolgende Tabelle 6.4 zeigt die ausgewählten Sets und die Anzahl ihrer identifizierten Variabilitätsbeziehungen.

Top 20	**Set A**	**Set B**	**Set C**	**Set D**
Relationsgruppen	13	27	17	14
Alternativgruppen	1	0	1	0
Optionale Konfigurationsopt.	60	39	50	48

Tabelle 6.4: Analysierte Sets und die Anzahl ihrer Variabilitätsbeziehungen

In Tabelle 6.4 sind für jedes Set die Anzahl der enthaltenen Relationsgruppen, Alternativgruppen und optionaler Konfigurationsoptionen aufgeführt, welche alle Kandidaten für unnötige Variabilität des jeweiligen Sets umfassen. Für diese können im Rahmen der Fallstudie B Restrukturierungsempfehlungen abgeleitet werden.

Methodik

Zur Vorbereitung der Fallstudie haben wir zunächst gemeinsam mit einigen der beteiligten Domänenexperten zehn verschiedene *Geschäftsregeln* spezifiziert und in einem Regelkatalog (Tabelle A.2) zusammengetragen. Zudem haben wir gemeinsam

eine entsprechende *Graphbeschreibung* (vgl. Unterabschnitt 4.2.1) definiert. Für die Durchführung der Fallstudie haben sich die Experten dann zur Anwendung der Geschäftsregeln `GR1` bis `GR5` aus dem Regelkatalog entschieden, da die Berücksichtigung der Regeln `GR6` bis `GR10` für die Fallstudie nicht möglich war. Hierfür wären zusätzliche Daten über die *Funktionen*, den *Strategischen Fit*, die *TCO* und die *Restrukturierungskosten* der Komponenten in den analysierten TAPs erforderlich gewesen, welche aber nicht für die automatisierte Verarbeitung zur Verfügung standen. Für die Graphbeschreibung haben sich die Domänenexperten dazu entschieden, alle Layer und Tiers sowie alle Komponenten mit einer Nutzungshäufigkeit von über 30% zu berücksichtigen. Zudem wurden die Kategorien *Applikationsserver* und *Datenbank* als Kerntechnologie definiert.

Für die Fallstudie haben wir dann den vorgeschlagenen Ansatz zur Ableitung von Restrukturierungsempfehlungen mit Hilfe unseres Softwareprototypen auf Basis der zuvor erstellten 150%-Modelle für die *Sets A–D* ausgeführt. Hierbei wurden die im Vorfeld spezifizierten Geschäftsregeln und die erstellte Graphbeschreibung angewendet. Obwohl die Komponenten in den betrachteten TAPs nicht in Bezug auf alle möglichen Kriterien aus dem Regelkatalog analysiert werden konnten, ist es dennoch möglich zu beurteilen, ob die daraus resultierenden Restrukturierungsempfehlungen für Experten hilfreich sind. Daher haben wir die beteiligten Domänenexperten gebeten, die automatisiert erstellten Empfehlungen für die beiden *Sets A* und *B* mit ihren Erwartungen zu vergleichen und die Eignung dieser Restrukturierungsempfehlungen für die Reduzierung der unnötigen Variabilität in den analysierten TAPs zu bewerten. Eine umfassende manuelle Analyse aller automatisiert abgeleiteten Restrukturierungsempfehlungen für jedes *Set A–D* war aufgrund der zeitlichen Beschränkungen der Experten nicht realisierbar.

Ergebnisse und Diskussion

Nachfolgend werden die Ergebnisse unserer Fallstudie dargestellt und im Hinblick auf die definierten Forschungsfragen diskutiert.

FF-B.1 – Allgemeiner Ansatz: Um zu bewerten, ob der vorgeschlagene Ansatz geeignet ist, Restrukturierungsempfehlungen abzuleiten, haben wir unseren Algorithmus nacheinander für die betrachteten Sets von ausgewählten TAPs ausgeführt. In der Tabelle 6.5 werden die Ergebnisse dieser Ausführungen dargestellt. Die Tabelle zeigt die Anzahl der abgeleiteten Ersetzungs- und Entfernungsempfehlungen für die einzelnen 150%-Modelle der jeweiligen *Top20*-Mengen aus den *Sets A–D*.

Top 20	Set A	Set B	Set C	Set D
Ersetzungsempfehlungen	15	29	36	13
Entfernungsempfehlungen	16	21	18	7

Tabelle 6.5: Anzahl abgeleiteter Restrukturierungsempfehlungen für die Sets A–D

Bei Betrachtung von *Set A* lässt sich feststellen, dass unser Algorithmus insgesamt 31 Restrukturierungsempfehlungen generiert hat. Hiervon sind 15 Ersetzungsempfehlungen für die 14 Variabilitätsgruppen dieses Sets abgeleitet worden. Darüber hinaus sind für das *Set B* 29 Ersetzungsempfehlungen für 27 Variabilitätsgruppen, für das *Set C* 36 Ersetzungsempfehlungen für 18 Variabilitätsgruppen und für das *Set D* 13 Ersetzungsempfehlungen für 14 Variabilitätsgruppen identifiziert worden. Für die drei *Sets A, B* und *D* wurde durchschnittlich etwa eine Ersetzungsempfehlung pro Variabilitätsgruppe abgeleitet. Für das *Set C* hingegen konnte diese Anzahl sogar verdoppelt werden. Obwohl die drei *Sets A, C* und *D* eine vergleichbare Anzahl von Variabilitätsgruppen enthalten, ist die Anzahl der abgeleiteten Ersetzungsempfehlungen für das *Set C* signifikant höher als für die *Sets A* und *D*. Dies zeigt, dass im *Set C* ein deutlich höheres Potential zur Reduzierung unnötiger Variabilität steckt. Der Grund hierfür ist auf der einen Seite, dass im Rahmen der Businessanalyse mehr höherwertige Komponenten in *Set C* als in den anderen beiden Sets bestimmt worden sind. Dies spricht dafür, dass diese Elemente den spezifizierten Geschäftsanforderungen besser entsprechen, woraus sich mehr Ersetzungsempfehlungen ableiten lassen. Auf der anderen Seite wurden im *Set C* durch die Technologieanalyse eine hohe Ähnlichkeit der jeweiligen VKGs identifiziert, sodass nur wenige Ersetzungspotentiale verworfen werden mussten. Dies lässt darauf schließen, dass die in *Set C* enthaltenen TAPs eine hohe technischen Ähnlichkeit aufweisen.

In Bezug auf die Entfernungsempfehlungen konnte unser Algorithmus für *Set A* 16 solcher Empfehlungen für die in *A* enthaltenen 60 unabhängigen optionalen Konfigurationsoptionen ableiten. Zudem wurden für *Set B* 21 Entfernungsempfehlungen für 39 optionale Konfigurationsoptionen, für *Set C* 18 Entfernungsempfehlungen für 50 optionale Konfigurationsoptionen und für *Set D* 7 Entfernungsempfehlungen für 48 optionale Konfigurationsoptionen identifiziert. Durch den Vergleich dieser Werte kann für solche Entfernungsempfehlungen kein Trend erkannt werden. Während für das *Set D* nur für etwa jede neunte optionale Konfigurationsoption eine entsprechende Entfernungsempfehlung ausgegeben wurde, ist dies im *Set B* für mehr als jede zweite geschehen. Die Werte der anderen beiden Sets liegen dazwischen. Der Hauptgrund dafür ist die unterschiedliche Anzahl an optionalen Konfigurationsoptionen mit einer Nutzungshäufigkeit von weniger als 10%, da nur für diese entsprechende Entfernungsempfehlungen auf Basis der spezifizierten Geschäftsregel `GR4` abgeleitet wurden. Daraus lässt sich schließen, dass beispielsweise im *Set B* deutlich mehr unabhängige Konfigurationsoptionen als im *Set D* enthalten sind, sodass hierfür auch mehr Entfernungsempfehlungen abgeleitet worden sind. So kann davon ausgegangen werden, dass sich im *Set B* eine größere Anzahl technischer Insellösungen befindet, welche von Experten manuell überprüft werden sollten.

Insgesamt lässt sich festhalten, dass unser vorgeschlagener Ansatz geeignet ist, um Restrukturierungsempfehlungen für unterschiedliche Sets von technisch verwandten TAPs automatisiert abzuleiten. Somit können wir die Forschungsfrage *FF-B.1* positiv beantworten.

FF-B.2 – Restrukturierungsempfehlungen: Nach der Ausführung unseres Algorithmus für die vier betrachteten Sets haben wir den an der Fallstudie beteiligten Experten die abgeleiteten Restrukturierungsempfehlungen für das *Set A* und das *Set B* präsentiert, um diese im Hinblick auf ihre Eignung zur Reduzierung von unnötiger Variabilität zu bewerten. Aus Zeitgründen waren unsere Experten nicht in der Lage, jede einzelne Empfehlung aller Sets zu bewerten. Jedoch haben sie sich einen Überblick über die abgeleiteten Restrukturierungsempfehlungen aller Sets verschafft und die Empfehlungen der beiden *Sets A* und *B* im Detail bewertet.

Nach Einschätzung der Experten sind die präsentierten Restrukturierungsempfehlungen gut geeignet, um auf konkrete Restrukturierungsmöglichkeiten zur Verringerung unnötiger Variabilität aufmerksam zu machen. Dabei erwähnten sie auch, dass die Qualität der vorgelegten Empfehlungen von den ausgewählten Geschäftsregeln und ihren Kriterien abhängt, sodass diese sorgfältig spezifiziert werden müssen. So haben die Experten auch einige Empfehlungen identifiziert, die sie als nicht sinnvoll erachtet haben. Zum Beispiel bestimmten sie zwei Empfehlungen, die vorschlagen eine Komponente zu ersetzen, welche strategisch bevorzugt ist. Darüber hinaus haben sie drei weitere Ersetzungsempfehlungen identifiziert, die den Austausch einer Komponente vorschlagen, welche erforderliche Funktionen für die Technologiearchitektur des betrachteten Systems zur Verfügung stellt. Die Ableitung solcher ungeeigneten Restrukturierungsempfehlungen ist dadurch zu erklären, dass sie ohne zusätzliche Informationen zum *Strategischen Fit* oder den *Funktionen* der einzelnen Komponenten generiert wurden, da diese für unsere Fallstudie nicht verfügbar waren. Wir sind jedoch überzeugt, dass die automatisierte Ableitung von Restrukturierungsempfehlungen präziser wird, wenn solche zusätzlichen Informationen bereitgestellt werden, um sie im Rahmen der Businessanalyse berücksichtigen zu können.

Nichtsdestotrotz bewerteten die Experten die Mehrzahl aller präsentierten Restrukturierungsempfehlungen als sinnvoll und hilfreich für die Architekturarbeit in ihrem industriellen Kontext. So konnten die Experten beispielsweise auch Restrukturierungspotentiale erkennen, die sie zuvor nicht erwartet hatten. Insgesamt waren die Experten überzeugt, dass die abgeleiteten Restrukturierungsempfehlungen hilfreich sind, um unnötige Variabilität zu identifizieren und zu reduzieren. Daher können wir ebenfalls die Forschungsfrage *FF-B.2* bestätigen.

6.2.4 Fallstudie C: Bewertung und Restrukturierungsentscheidung

Im Kontext der Fallstudie C stand die Evaluierung unseres automatisierten Ansatzes zur Bewertung von Restrukturierungsempfehlungen und der Unterstützung von Restrukturierungsentscheidungen im Fokus (vgl. Kapitel 5). Zu diesem Zweck wurden die folgenden Forschungsfragen (FF) formuliert:

FF-C.1 – Delta Analyse: *Ist die entworfene Delta Sprache geeignet, um das resultierende 150%-Modell für konkrete Restrukturierungsempfehlungen zu simulieren?*

FF-C.2 – Abhängigkeitsanalyse: *Ist die vorgeschlagene Anpassungsbedarfsanalyse geeignet, um Anpassungsabhängigkeiten einer konkreten Restrukturierungsempfehlung zu identifizieren?*

FF-C.3 – Aufwandsschätzung: *Ist der vorgeschlagene Ansatz zur Aufwandsschätzung geeignet, um die erforderlichen Aufwände einer konkreten Restrukturierungsempfehlungen abzuschätzen?*

FF-C.4 – Variabilitätsreduzierung: *Ist der vorgeschlagene iterative Ansatz geeignet, um Experten bei der Variabilitätsreduzierung zu unterstützen?*

Für diese Fallstudie standen ebenfalls drei Domänenexperten unseres Industriepartners zur Verfügung. Zur Durchführung der Fallstudie haben wir wieder die *Top20*-Mengen der *Sets A–D* verwendet und diese mit Hilfe unseres Softwareprototypen verarbeitet. Dabei haben wir auf die bereits vorgestellten Variabilitätsinformationen (vgl. Tabelle 6.4) und die daraus abgeleiteten Restrukturierungsempfehlungen (vgl. Tabelle 6.5) für die *Sets A–D* zurückgegriffen.

Methodik

Zur Vorbereitung der Fallstudie haben wir zunächst zusammen mit den beteiligten Experten ein passendes *Abhängigkeitsmodell* (vgl. Unterabschnitt 5.1.2) und eine entsprechende *Aufwandsmatrix* (vgl. Unterabschnitt 5.1.3) spezifiziert. Danach haben wir dann alle 155 Restrukturierungsempfehlungen mit Hilfe unseres Softwareprototypen bewertet. Zur Beantwortung der Forschungsfrage *FF–C.1* haben wir anschließend jedes Ist-150%-Modell der vier betrachteten *Sets A–D* für alle bewerteten Restrukturierungsempfehlungen des jeweiligen Sets transformiert, um das daraus resultierende Soll-150%-Modell mit den Deltaoperationen der entsprechenden Restrukturierungsempfehlung vergleichen zu können. So konnten wir analysieren, ob die generierten Soll-150%-Modelle korrekt transformiert wurden. Anschließend haben die beteiligten Experten jeweils fünf Restrukturierungsempfehlungen für jedes Set nach dem Zufallsprinzip ausgewählt, um deren Bewertung im Detail zu analysieren und damit die Forschungsfragen *FF–C.2* und *FF–C.3* beantworten zu können. Zu diesem Zweck haben wir die Experten gebeten, die generierten Bewertungsergebnisse der Anpassungsbedarfsanalyse sowie der Aufwandsschätzung für jede der ausgewählten Restrukturierungsempfehlungen manuell zu analysieren und zu verifizieren. Aufgrund der zeitlichen Beschränkung der Experten war eine umfassende Analyse der Bewertungsergebnisse aller Empfehlungen aus jedem der vier *Sets A–D* nicht möglich. Wir sind jedoch überzeugt, dass die Anzahl der evaluierten Abhängigkeiten und Aufwandswerte ausreichend ist, um die Eignung unseres Ansatzes zu zeigen. Abschließend haben wir dann die Experten darum gebeten, konkrete Restrukturierungsentscheidungen für die vier betrachteten *Sets A–D* mit Hilfe unseres Softwareprototypen zu treffen. So waren wir in der Lage, auch die Eignung unseres iterativen Ansatzes zur Entscheidungsunterstützung für die Reduzierung der Variabilität in Bezug auf die Forschungsfrage *FF–C.4* zu bewerten.

Ergebnisse und Diskussion

Im Folgenden zeigen wir die Ergebnisse unserer Fallstudie und diskutieren diese im Hinblick auf die formulierten Forschungsfragen.

FF-C.1 – Delta Analyse: Zur Beantwortung dieser Forschungsfrage haben wir die Ist-150%-Modelle aus den jeweiligen *Sets A–D* für alle ausgewählten Restrukturierungsempfehlungen mit Hilfe ihrer Deltaoperationen transformiert. In der folgenden Tabelle 6.6 zeigen wir die Ergebnisse für diese automatisierten Transformationen.

	Set A		**Set B**		**Set C**		**Set D**	
	+	–	+	–	+	–	+	–
Ersetzungsempfehlungen	15	0	29	0	36	0	13	0
Entfernungsempfehlungen	16	0	21	0	18	0	7	0

Anzahl korrekt (+) und inkorrekt (–) transformierter 150%-Modelle

Tabelle 6.6: Ergebnisse der 150%-Modell-Transformationen für die ausgewählten Restrukturierungsempfehlungen

Durch die manuelle Überprüfung aller resultierenden Soll-150%-Modelle konnten wir analysieren, ob die einzelnen Restrukturierungsempfehlungen korrekt umgesetzt worden sind. Als Ergebnis zeigt die Tabelle 6.6 die Anzahl der *korrekt (+)* und *inkorrekt (–)* transformierten Soll-150%-Modelle. Wie hier zu erkennen ist, wurde das jeweilige Soll-150%-Modell in allen Fällen korrekt transformiert. Somit konnten alle ausgewählten Ersetzungs- und Entfernungsempfehlung erwartungsgemäß simuliert werden. Daraus lässt sich schließen, dass die entworfene Delta Sprache geeignet ist, um die erforderlichen Deltaoperationen für jede einzelne Restrukturierungsempfehlung zu generieren und somit die jeweiligen 150%-Modelle korrekt zu transformieren. Daher kann die Forschungsfrage *FF–C.1* positiv beantwortet werden.

FF-C.2 – Abhängigkeitsanalyse: Um diese Forschungsfrage beantworten zu können, haben wir die an der Fallstudie beteiligten Experten gebeten, fünf Restrukturierungsempfehlungen je *Set A–D* zufällig auszuwählen und deren automatisch identifizierte Abhängigkeiten manuell zu überprüfen. Die folgende Tabelle 6.7 zeigt die Evaluationsergebnisse für diese manuelle Überprüfung. Hierbei wird für jedes einzelne *Set A–D* angegeben, wie hoch die Anzahl der *automatisch (A)* identifizierten Abhängigkeiten (N_A) der jeweiligen Restrukturierungsempfehlungen ist. Im Vergleich dazu ist ebenfalls beschrieben, wie viele dieser Abhängigkeiten *manuell (M)* durch die Experten verifiziert worden sind (N_M). Stimmen diese beiden Werte für eine gegebene Restrukturierungsempfehlung `REi` überein, gilt also $\mathrm{N_A}(\mathrm{REi}) = \mathrm{N_M}(\mathrm{REi})$, so liegt eine 100-prozentige Übereinstimmung zwischen den automatisch identifizierten und den manuell verifizierten Abhängigkeiten für diese Empfehlung vor. Solche Werte

	Set A		Set B		Set C		Set D	
	A	**M**	**A**	**M**	**A**	**M**	**A**	**M**
1. Empfehlung	16	1	13	11	**24**	**24**	11	9
2. Empfehlung	11	9	6	5	20	15	11	6
3. Empfehlung	10	6	**3**	**3**	9	7	4	2
4. Empfehlung	**6**	**6**	**2**	**2**	**8**	**8**	**3**	**3**
5. Empfehlung	**1**	**1**	**1**	**1**	6	4	**1**	**1**
∅ **Übereinstimmung**	**ca. 70%**		**ca. 94%**		**ca. 84%**		**ca. 77%**	

Anzahl automatisch (A) identifizierter und manuell (M) verifizierter Abhängigkeiten

Tabelle 6.7: Ergebnisse für die Verifikation von identifizierten Abhängigkeiten

sind in der Tabelle 6.7 hervorgehoben. Weiterhin ergibt sich eine anteilige Übereinstimmung aus dem Verhältnis $\frac{N_M(REi)}{N_A(REi)}$. Durch die Bestimmung des Mittelwertes für alle anteiligen Übereinstimmungen der fünf Restrukturierungsempfehlungen eines Sets ergibt sich so die durchschnittliche Übereinstimmung für dieses Set.

Die minimale durchschnittliche Übereinstimmung beträgt 70% für das *Set A* und die maximale 94% für das *Set B*. Dabei ist zu sehen, dass die automatisch generierten Abhängigkeiten von mindestens zwei Restrukturierungsempfehlungen in jedem Set zu 100% verifiziert wurden. Im *Set B* mit der höchsten durchschnittlichen Übereinstimmung trifft dies sogar auf 3 Empfehlungen zu. Dabei lässt sich auch erkennen, dass solche 100% Übereinstimmungen eher bei Restrukturierungsempfehlungen mit einer geringeren Anzahl an Abhängigkeiten festgestellt werden konnte. Von den insgesamt 9 Übereinstimmungen sind nur drei Empfehlungen mit mehr als fünf Abhängigkeiten zu verzeichnen. Nichtsdestotrotz lässt sich auch für die übrigen Restrukturierungsempfehlungen eine hohe Übereinstimmung zwischen automatisch identifizierten und manuell verifizierten Abhängigkeit feststellen. Dies spiegelt sich auch in den hohen Durchschnittswerten für die Übereinstimmung in den jeweiligen *Sets A–D* wieder.

Lediglich für eine einzelne Restrukturierungsempfehlung konnten die identifizierten Abhängigkeiten nicht verifiziert werden. Dabei handelt es sich um die *1. Empfehlung* im *Set A*, für die 16 Abhängigkeiten automatisch bestimmt worden sind, aber nur eine einzige von Experten bestätigt werden konnte. Der Grund hierfür ist der spezifische Architekturtyp des betroffenen Softwaresystems, für den unser Abhängigkeitsmodell keine passenden abstrakten Abhängigkeiten enthielt. Um ein solches Problem zu umgehen, können typspezifische Abhängigkeitsmodelle (z.B. für *Rich-Client-* und *Web-Client-Anwendung*) erstellt und die ausgewählten TAPs nach dem jeweiligen Architekturtyp kategorisiert werden. So können TAPs vom gleichen Architekturtyp zu einem Set zusammengestellt werden, was zu präziseren Ergebnissen in der Abhängigkeitsanalyse führt. Obwohl dazu initial ein höherer Aufwand für die Erstellung

typspezifischer Abhängigkeitsmodelle erforderlich ist, können solche spezifischen Abhängigkeitsmodelle wiederverwendet werden und damit langfristig zu einer besseren Unterstützung von Domänenexperten für die Reduzierung von Variabilität führen.

Insgesamt kann durch die Verwendung unseres Ansatz für die Abhängigkeitsanalyse eine durchschnittliche Gesamtübereinstimmung von 81% über alle vier *Sets A–D* festgestellt werden. Da sich diese zudem durch die Verwendung von typspezifischen Abhängigkeitsmodellen potentiell weiter erhöhen lässt, kann auch die Forschungsfrage *FF-C.2* positiv beantwortet werden.

FF-C.3 – Aufwandsschätzung: Für diese Forschungsfrage haben wir die beteiligten Experten gebeten, die automatisch ermittelten Migrations- und Anpassungsaufwände für jede der ausgewählten 20 Restrukturierungsempfehlungen (5 pro *Set A–D*) durch eine manuelle Überprüfung zu verifizieren. Hierfür haben wir die jeweiligen Migrationsaufwände und Anpassungsaufwände getrennt von einander analysiert. In Tabelle 6.8 sind die *automatisch (A)* bestimmten sowie die *manuell (M)* verifizierten *Migrationsaufwände* dargestellt. Beispielsweise wurde für die erste Restrukturierungsempfehlung im *Set A* ein Migrationsaufwand von 12 automatisch (A) bestimmt und anschließend auch manuell (M) in dieser Höhe bestätigt.

Wie in Tabelle 6.8 zu erkennen ist, konnten alle automatisiert bestimmten Migrationsaufwände durch die an der Fallstudie beteiligten Experten verifiziert werden. Daraus ergibt sich eine 100-prozentige Übereinstimmung der Migrationsaufwände für die ausgewählten Restrukturierungsempfehlungen in allen betrachteten Sets A–D. Hinsichtlich der Anpassungsaufwände stellt sich ein etwas anderes Bild dar, wie die folgende Abbildung 6.9 zeigt. Auch hier sind die automatisch (A) bestimmten Aufwände im Vergleich zu den manuell (M) verifizierten Aufwänden dargestellt. Von den 20 analysierten Restrukturierungsempfehlungen konnten die Anpassungsaufwände von 7 dieser Empfehlungen durch unsere Experten in gleicher Höhe verifiziert werden.

	Set A		**Set B**		**Set C**		**Set D**	
	A	**M**	**A**	**M**	**A**	**M**	**A**	**M**
1. Empfehlung	12	12	2	2	12	12	3	3
2. Empfehlung	6	6	12	12	12	12	12	12
3. Empfehlung	3	3	4	4	12	12	4	4
4. Empfehlung	14	14	4	4	21	21	8	8
5. Empfehlung	3	3	3	3	6	6	6	6
∅ Übereinstimmung	**100%**		**100%**		**100%**		**100%**	

automatisch (A) bestimmte und manuell (M) verifizierte Migrationsaufwände

Tabelle 6.8: Verifikationsergebnisse für die geschätzten Migrationsaufwände

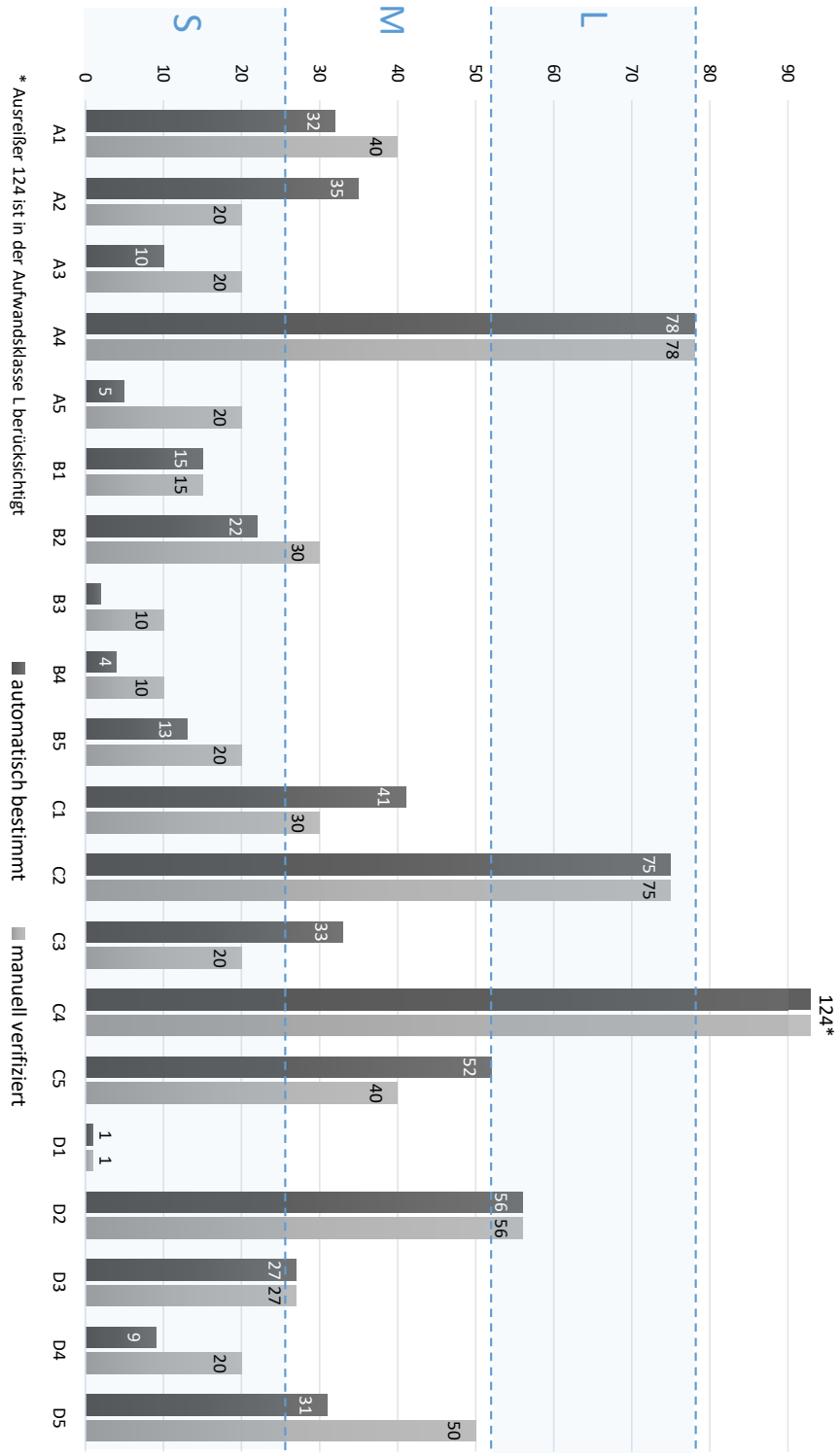


Abbildung 6.9: Verifikationsergebnisse für die geschätzten Anpassungsaufwände

Bei den anderen 13 Restrukturierungsempfehlungen haben die Experten abweichende Werte für die jeweiligen Anpassungsaufwände manuell bestimmt. Dabei wurden die automatisiert bestimmten Werte in 9 Fällen nach oben und in 4 Fällen nach unten korrigiert. Auffällig ist, dass in 5 von den 9 nach oben korrigierten Fällen der jeweilige Aufwandswert auf die nächstmögliche durch zehn teilbare Zahl angehoben wurde, also beispielsweise von 32 auf 40 (`A1`) oder von 4 auf 10 (`B4`). Dies lässt sich darauf zurückführen, dass die Experten bei der manuellen Aufwandsschätzung eine grobkörnigere Skala in 10er-Schritten angewendet haben. Ein direkter Vergleich der automatisiert und manuell ermittelten Aufwandswerte ist somit nicht zielführend. Beispielsweise liegt der automatisiert bestimmte Anpassungsaufwand für `B3` bei 2 und der dazugehörige manuelle Wert bei 10. Dieser Wert ist zwar um das 5-fache höher als der automatisiert ermittelte, stellt jedoch den niedrigsten manuell geschätzten Aufwandswert dar.

Um die automatisiert bestimmten und die manuell verifizierten Anpassungsaufwände besser vergleichbar zu machen, haben wir daher die drei *Aufwandsklassen S, M, L* gebildet. Diese stellen, ähnlich wie die entsprechenden Klassen für die Aufwandsmatrix (vgl. Unterabschnitt 5.1.3), Kategorien mit *geringem (S)*, *mittelmäßigen (M)* oder *hohem (L)* Anpassungsaufwand dar. Diese Aufwandsklassen haben wir auf Basis der verfügbaren Aufwandswerte gleich verteilt, beginnend mit dem Wert 1 (für `D1`) und endet mit dem Wert 78 (für `A4`). Den Wert 124 für `C4` betrachten wir dabei als Ausreißer, weshalb wir die entsprechende Restrukturierungsempfehlung ebenfalls in der Aufwandsklasse L berücksichtigen. So ergeben sich die drei gleich verteilten Wertbereiche *S=1–26*, *M=27–52* und *L=53–78 (inkl. 124)*.

Die Tabelle 6.9 zeigt die Anzahl der entsprechenden Restrukturierungsempfehlungen in der jeweiligen Aufwandsklasse. Wie hier zu sehen ist, sind 9 dieser Empfehlungen aufgrund ihrer automatisiert bestimmten Anpassungsaufwände der Klasse S, 7 weitere der Klasse M und die 4 übrigen der Klasse L zugeordnet worden. Nach der manuellen Überprüfung sind 8 Restrukturierungsempfehlungen (89%) in der Klasse S, 5 (71%) in der Klasse M und alle 4 (100%) in der Klasse L verblieben. Daraus ergibt

	Aufwandsklassen für Anpassungsaufwände		
	S (1–26)	**M (27–52)**	**L (53–78)***
Anzahl REs (automatisch bestimmt)	9	7	4
Anzahl REs (manuell verifiziert)	8	5	4
Übereinstimmung	**ca. 89%**	**ca. 71%**	**100%**

** beinhaltet den Ausreißer 124*

Tabelle 6.9: Anzahl der Restrukturierungsempfehlungen (REs) in den drei Aufwandsklassen S, M und L

sich eine durchschnittliche Übereinstimmung von 87% über alle Aufwandsklassen.

Insgesamt lässt sich damit feststellen, dass sowohl die Ergebnisse für die Schätzung des Migrationsaufwandes mit 100% Übereinstimmung als auch die Ergebnisse für die Ermittlung der Anpassungsaufwände mit durchschnittlich 87% Übereinstimmung für die beteiligten Experten sehr zufriedenstellend sind. Insofern kann die Forschungsfrage *FF-C.3* positiv beantwortet werden.

FF-C.4 – Variabilitätsreduzierung: Um die letzte Forschungsfrage zu beantworten, haben wir unsere Experten gebeten, das jeweilige 150%-Modell der vier *Sets A–D* mit Hilfe unseres Softwareprototypen und den generierten Restrukturierungsempfehlungen (vgl. Tabelle 6.5) zu restrukturieren. Hierzu haben wir sie mit den Ergebnissen der Portfoliobewertung und der Simulationsmöglichkeit vertraut gemacht. Diese sollten die Experten nutzen, um konkrete Restrukturierungsentscheidungen für die vier *Sets A–D* zu treffen. Als Ergebnis wurden die von den Experten generierten Soll-150%-Modelle mit den entsprechenden Ist-150%-Modellen verglichen und hinsichtlich der erzielten Variabilitätsreduktion analysiert. In Tabelle 6.10 stellen wir die Ergebnisse dieser Analyse dar.

Die Tabelle 6.10 enthält für alle *Sets A–D* die Anzahl der Variabilitätsgruppen (Alternativ- und Relationsgruppen) sowie die Anzahl der alternativen und optionalen Konfigurationsoptionen für das jeweilige Ist-150%-Modell (M_1) und das entsprechende Soll-150%-Modell (M_2). Außerdem werden die ermittelten Werte für die *Variabilitätsdifferenzen (VD)* gemäß der Gleichung 5.1 in absoluten und relativen Zahlen angegeben. Hinsichtlich der Variabilitätsgruppen waren die Experten durch die Unterstützung unseres Ansatzes in der Lage, ihre Anzahl um mindestens -14.3% (*Set D*), um höchstens -25.9% (*Set B*) und im Durchschnitt um -21.0% zu

	Variabilitäts-gruppen (VG)			**Alternative (A) Konfigurationsopt.**			**Optionale (O) Konfigurationsopt.**			
	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{VD_{VG}}$	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{VD_A}$	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{VD_O}$	$\mathbf{VD_G}$
Set A	14	11	-3 -21.4%	6	4	-2 -33.3%	95	85	-10 -10.5%	-15 -13.0%
Set B	27	20	-7 -25.9%	0	0	0	105	93	-12 -11.4%	-19 -14.4%
Set C	18	14	-4 -22.2%	3	2	-1 -33.3%	110	99	-11 -10.0%	-16 -12.2%
Set D	14	12	-2 -14.3%	0	0	0	99	90	-9 -9.1%	-11 -9.7%
∅	**-21,0%**			**-33,3%***			**-10,3%**			**-12.3%**

*VD_G = Gesamtvariabilitätsdifferenz = ΔVariabilität(M_1,M_2) * berücksichtigt nicht die leeren Sets B und D*

Tabelle 6.10: Erzielte Variabilitätsreduktionen für die Sets A–D

reduzieren. Zudem konnten auch die alternativen Konfigurationsoptionen im *Set A* und im *Set C* um jeweils 33,3% verringert werden. Der Großteil aller variablen Elemente in den analysierten TAPs waren optionale Konfigurationsoptionen. Diese konnten um mindestens -9,1% (*Set D*), um höchstens -11,4% (*Set B*) und durchschnittlich um -10,3% reduziert werden. Insgesamt erreichten die Experten durch die Verwendung unseres Ansatz eine durchschnittliche Variabilitätsreduzierung von -12,3% über alle *Sets A–D*.

Die Gründe für diese Ergebnisse sind vielfältig. Zum Beispiel ist eine große Anzahl der optionalen Konfigurationsoptionen unabhängig und damit keiner Variabilitätsgruppe zugeordnet. Daher gibt es für diese oft keinen äquivalenten Ersatz, der stattdessen verwendet werden könnte. Somit konnten für die betroffenen Komponenten auch nicht viele Ersetzungsempfehlungen abgeleitet werden. Zudem weisen viele dieser optionalen Elemente auch eine höhere Nutzungshäufigkeit von mehr als 20% auf. Dadurch können sie nicht mehr als Einzellösung betrachtet werden, sodass auch keine entsprechende Entfernungsempfehlung generiert wird. Daher ist eine starke Reduzierung der hohen Anzahl von optionalen Konfigurationsoptionen automatisiert kaum möglich. Hier bedarf es weiterhin der manuellen Analyse von Experten mit ihrem Domänenwissen. Darüber hinaus konnten in vielen Fällen nicht alle Varianten innerhalb der gleichen Variabilitätsgruppe ersetzt werden, sodass für diese nur eine standardisierte Komponente übrig bleibt. Als Grund dafür gaben die Experten an, dass sie bewusst eine Multi-Lieferanten-Strategie anvisieren, um nicht in die Abhängigkeit von einem einzigen Hersteller für eine bestimmte Kategorie von Technologieprodukten zu geraten. Außerdem haben die Experten erkannt, dass die Funktionen der betrachteten Komponenten innerhalb einer Variabilitätsgruppe nicht immer deckungsgleich sind. Solche Komponenten gegeneinander auszutauschen, macht mit Blick auf den spezifischen Anwendungsfall der betroffenen Softwaresysteme nicht immer Sinn. Somit ist es auch kaum möglich, alle Varianten derselben Variabilitätsgruppe zu eliminieren. Verbessern ließe sich diese Situation mit einem detaillierteren Kategoriensystem, welches Komponenten noch präziser anhand ihrer Funktionen klassifiziert. Dies bedeutet aber wiederum eine höheren manuellen Aufwand für die Erstellung solch eines Systems und die Zuordnung aller Komponenten zu den neu erstellten Kategorien. Auf lange Sicht würde sich dieser Aufwand aber auszahlen, da somit präzisere Ergebnisse im Rahmen der automatisierten Analyse erreicht werden können und die manuelle Überprüfung dieser Ergebnisse weniger Zeit in Anspruch nimmt.

Nichtsdestotrotz konnten die an der Fallstudie beteiligten Domänenexperten die Variabilität technisch verwandter Softwaresysteme mit Hilfe unseres Ansatzes deutlich reduzieren. Obwohl die Gesamtreduktion der Variabilität von -12,3% für die betrachteten TAPs unserer Fallstudie vermeintlich gering erscheint, sind die Experten überzeugt, dass selbst dieses Ergebnis einen effektiven Beitrag zur Reduzierung der Kosten für Wartung und Weiterentwicklung sowie zur Verbesserung der Anpassungsfähigkeit an neue Rahmenbedingungen führt. Dies wiederum hilft dem Unternehmen, die

eigenen Ziele besser zu erreichen. Somit können wir auch die letzte Forschungsfrage *FF-C.4* positiv beantworten.

6.3 Experteninterviews

Um unseren Ansatz zur Reduzierung von Variabilität auch im Hinblick auf dessen Verständlichkeit und Nützlichkeit zu evaluieren, haben wir individuelle Interviews mit verschiedenen Experten unseres Industriepartners durchgeführt. Dabei zielen diese Experteninterviews darauf ab, die folgenden Forschungsfragen (FF) zu beantworten:

FF-I.1 – **Verständlichkeit:** *Ist der vorgeschlagene Ansatz verständlich und anwendbar für Experten?*

FF-I.2 – **Nützlichkeit:** *Sind die generierten Ergebnisse nützlich und hilfreich für Experten?*

Die Teilnehmer der Expertenbefragung werden im Unterabschnitt 6.3.1 kurz vorgestellt, bevor die Methodik für die Interviews im Unterabschnitt 6.3.2 erläutert und deren Ergebnisse im Unterabschnitt 6.3.3 präsentiert und diskutiert werden.

6.3.1 Teilnehmer der Expertenbefragung

Im Rahmen der durchgeführten Experteninterviews wurden insgesamt sechs unterschiedliche IT-Architekten unseres Industriepartners befragt. Bei der Auswahl dieser Experten wurden sowohl deren Berufserfahrung als auch deren Tätigkeitsschwerpunkt berücksichtigt. Um hierbei eine möglichst gute Streuung zu erreichen, sind IT-Architekten mit einer Berufserfahrung zwischen 2,5 und 10 Jahren sowie mit unterschiedlichen Tätigkeitsschwerpunkten (technisch, fachlich, Projekt- und Managementlevel) berücksichtigt worden.

Die folgende Tabelle 6.11 listet die befragten Experten zusammen mit ihrer Berufserfahrung und der Beschreibung ihrer Tätigkeit auf.

Experte	Job Beschreibung	Berufserfahrung
A	Projekt IT Architekt	10 Jahre
B	Enterprise Architekt	9 Jahre
C	IT Technologie Architekt	6 Jahre
D	Teamleiter Enterprise Architecture Management	5 Jahre
E	Domänenarchitekt	3 Jahre
F	Enterprise Technologie Architekt	2,5 Jahre

Tabelle 6.11: Teilnehmer der Expertenbefragung

6.3.2 Methodik der Expertenbefragung

Die Interviews wurden mit jedem Experten einzeln durchgeführt. Dabei waren die gestellten Fragen den Experten im Vorfeld nicht bekannt. Gleiches gilt für die Antworten anderer Experten. Vor Beginn der eigentlichen Interviews wurde den Experten eine Einführung in die vorgeschlagenen Techniken unseres Ansatzes gegeben und beispielhafte Artefakte (z.B. ein spezifisches 150%-Modell, entsprechende Restrukturierungsempfehlungen sowie deren Abhängigkeiten und Aufwände) anhand unseres Softwareprototypen präsentiert. So konnten sich die Experten mit unseren Techniken vertraut machen und alle Unklarheiten beseitigt werden.

Fragebogen: Der erstellte Fragebogen enthält insgesamt 13 offene Fragen, welche die Verständlichkeit der vorgeschlagenen Methoden sowie die Nützlichkeit der daraus resultierenden Ergebnisse für die Experten fokussieren. Diese Aspekte wurden jeweils im Kontext der drei Bereiche *Variabilitätsanalyse* (vgl. Kapitel 3), *Ableitung von Restrukturierungsempfehlungen* (vgl. Kapitel 4) sowie *Bewertung und Restrukturierungsentscheidung* (vgl. Kapitel 5) untersucht. Der vollständige Fragebogen mit den Fragen für alle Bereiche kann dem Abschnitt A.5 im Anhang entnommen werden.

Interviews: Zur Vorbereitung der Interviews wurden 60 Minuten eingeplant, um die zu befragenden Experten mit den Techniken unseres vorgeschlagenen Ansatzes und dem entwickelten Softwareprototypen vertraut zu machen. Die einzelnen Interviews sind anschließend mit einer Dauer von 30–40 Minuten durchgeführt worden. Im Rahmen dessen wurde den Experten das 150%-Modell für die *Top20* TAPs aus dem *Set A* (vgl. Tabelle 6.4) zur Bewertung des ersten Bereiches *Variabilitätsanalyse* vorgestellt. Für die Beurteilung des zweiten Bereiches *Ableitung von Restrukturierungsempfehlungen* haben wir den Experten vier beispielhafte Geschäftsregeln aus unserem Regelkatalog, drei exemplarische Graphbeschreibungen sowie drei zufällig ausgewählte Restrukturierungsempfehlungen für das gezeigte 150%-Modell präsentiert. Im Anschluss haben wir den Experten für diese Restrukturierungsempfehlungen die Ergebnisse der Differenzanalyse, der Abhängigkeitsanalyse und der Aufwandsschätzung gezeigt, um auch eine Bewertung des dritten Bereiches *Bewertung und Restrukturierungsentscheidung* zu ermöglichen. Zudem haben wir die Simulation eines Soll-150%-Modells mit Hilfe unseres Softwareprototypen für die *Top20* TAPs exemplarisch gezeigt, um auch diese Technik von den Experten beurteilen zu lassen.

Unter Berücksichtigung dieser Artefakte wurden die Experten auf Grundlage des erstellten Fragebogens interviewt. Zur Auswertung wurden die entsprechenden Interviews danach transkribiert und anschließend einer qualitativen Inhaltsanalyse [May07] unterzogen. Die daraus gewonnen Ergebnisse wurden abschließend interpretiert.

6.3.3 Ergebnisse und Diskussion

Im Folgenden stellen wir die Ergebnisse der Experteninterviews für die drei benannten Bereiche vor und diskutieren diese.

Variabilitätsanalyse

FF-I.1 – Verständlichkeit: Allen interviewten Experten (100%) gefiel das Konzept von aggregierten Variabilitätsinformationen in einem einzigen Modell. Sie sagten aus, dass sie hierdurch schnell in der Lage sind, alle Gemeinsamkeiten und Unterschiede der analysierten TAPs zu erkennen. Fünf Experten (83%) hoben dabei die Visualisierung der spezifischen Variabilitätsbeziehungen von Komponenten und ihren Nutzungshäufigkeiten für unterschiedliche Versionen als sehr gut verständlich hervor. Insgesamt waren die Experten davon überzeugt, dass der Ansatz nachvollziehbar und gut anwendbar ist.

FF-I.2 – Nützlichkeit: Alle Experten (100%) haben berichtet, dass unser Variabilitätsmining-Ansatz einen Mehrwert gegenüber den vorhandenen Daten in den analysierten TAPs bietet, da es Architekten so ermöglicht wird, fundierte Restrukturierungsentscheidungen auf Basis von konkreten Variabilitätsinformationen zu treffen. Fünf Experten (83%) stellten heraus, dass durch die Visualisierung von Variabilitätsbeziehungen Details ersichtlich werden, die sonst nur mit viel manuellem Aufwand verfügbar gemacht werden könnten. Als Beispiel nennen vier Experten (66%) die Identifizierung von funktional ähnlichen Komponenten (z.B. verschiedene Datenbank- oder Betriebssysteme), welche zu unnötiger Variabilität führen, da sie einen sehr ähnlichen Verwendungszweck erfüllen. Daher bewerten die Experten ein 150%-Modell als hilfreiche Grundlage, um ähnliche Architekturlösungen zu bestimmen. Dies ermöglicht die Reduzierung des Aufwands und der Gesamtkosten für die Wartung und Weiterentwicklung solcher Lösungen durch die Standardisierung ausgewählter Komponenten. Drei Experten (50%) waren sich zudem einig, dass der Mehrwert unseres Ansatzes den initialen Aufwand für die damit verbundenen Tätigkeiten (z.B. Auswahl geeigneter TAPs) übersteigt, da hierdurch eine große Menge von TAPs gleichzeitig analysiert werden kann und so beispielsweise unternehmensweite Architekturrichtlinien entwickelt werden können. Darüber hinaus betonten zwei der Experten (33%), dass das 150%-Modell eine zusätzliche Unterstützung darstellt, um die technische Schuld in gewachsenen Architekturen zu erkennen (zum Beispiel aufgrund von Designentscheidungen, welche die Wartbarkeit negativ beeinflussen). Als Konsequenz können geeignete Gegenmaßnahmen eingeleitet und deren Auswirkungen überwacht werden. So konnte ein Experte (16%) beispielsweise auf einen Blick erkennen, dass eine kürzlich getroffene Strategieentscheidung zu Gunsten einer konkreten Technologie bereits positive Auswirkungen auf die analysierten TAPs hat.

Ableitung von Restrukturierungsempfehlungen

FF-I.1 – Verständlichkeit: Fast jeder Experte (83%) hat ausgesagt, dass der vorgeschlagene Ansatz zur Ableitung von Restrukturierungsempfehlungen verständlich und leicht anwendbar ist. Dabei haben die Experten hervorgehoben, dass es unerlässlich ist, eine technische Komponente aus verschiedenen Perspektiven zu untersuchen, um ein mögliches Restrukturierungspotential identifizieren zu können. In diesem Zusammenhang haben wir den Experten die vier Geschäftsregeln `GR1`, `GR4`, `GR6`, und `GR9` aus unserem Regelkatalog (vgl. Tabelle A.2) zur Bewertung vorgelegt. Fast alle Experten (83%) haben bestätigt, dass diese Geschäftsregeln verständlich sind und deren Anwendung dazu beitragen kann, eine entsprechende Komponente aus einer businessorientierten Sicht zu bewerten. So waren sie in der Lage, die Bedeutung der gezeigten Beispiel-Artefakte richtig zu bestimmen.

Zur Beurteilung der Technologieanalyse haben wir den Experten drei unterschiedliche Graphbeschreibungen präsentiert. Vier Experten (66%) sagten, dass die präsentierten Graphbeschreibungen verständlich sind. Die anderen beiden (33%) beschrieben, dass der `GIVEN`-Teil weniger selbsterklärend ist. Dies führen sie auf die Verknüpfung verschiedener Kriterien durch logische Operatoren (z.B. `AND`, `OR`) zurück. Um das Verständnis hierfür zu verbessern, kann eine noch genauere Anleitung zur Verwendung unserer DSL erfolgen sowie anschauliche Beispiele gegeben werden.

FF-I.2 – Nützlichkeit: Die meisten Experten (83%) erklärten, dass die vier gezeigten Geschäftsregeln nützlich sind, um die Geschäftsanforderungen und -ziele zu beschreiben und damit die Komponenten aus Businesssicht zu analysieren. Besonders hilfreich fanden fünf Experten (83%) dabei die Verwendung von `GR4` und `GR6` sowie vier Experten (66%) die Nutzung von `GR1`. Allerdings haben auch zwei der befragten Architekten (33%) ausgesagt, dass sie die in `GR1` gesetzte Schwelle für die Nutzungshäufigkeit von 33% als nicht hilfreich einschätzen. Aus diesem Grund haben wir eine weitere Geschäftsregel `GR2` mit einem Schwellwert von 66% für die Nutzungshäufigkeit in den Regelkatalog aufgenommen. So können Experten zwischen zwei unterschiedlichen Schwellwerten für die gleiche Regel wählen. Daraus lässt sich jedoch schließen, dass unterschiedliche Schwellwerte für die betrachteten Kriterien in den Geschäftsregeln benötigt werden. Dies lässt sich einfach umsetzten, indem die verfügbaren Regeln aus unserem Regelkatalog angepasst oder neue Geschäftsregeln mit Hilfe der entwickelten DSL spezifiziert werden. In diesem Zusammenhang haben drei Experten (50%) erwähnt, dass die Bestimmung von spezifischen Schwellwerten eine Herausforderung sein kann. Jedoch betrachten die Experten den Regelkatalog als hilfreiche Vorlage für die Entwicklung weiterer Geschäftsregeln. Darüber hinaus können Architekten auch durch eine zusätzliche Beratung dabei unterstützt werden, geeignete Schwellwerte für ihre Geschäftsregeln zu identifizieren.

Nach der Beurteilung der drei Graphbeschreibungen waren alle Experten (100%) überzeugt, dass die Analyse der Ähnlichkeit zweier Graphen sehr hilfreich ist, um die technische Machbarkeit eines Restrukturierungspotentials zu bestimmen. Außerdem bewerteten vier der Experten (66%) die Unterscheidung zwischen Kern- und

Unterstützungstechnologie als nützlich, da dieses Vorgehen geeignet ist, um die entsprechenden Kerntechnologieelemente eines TAPs separat zu betrachten. Die übrigen Experten (33%) waren nicht von der Nützlichkeit dieser Unterscheidung überzeugt. Aus diesem Grund bietet unser Ansatz den Experten die Möglichkeit, selbst zu entscheiden, ob sie diese Unterscheidung im Rahmen der Analyse berücksichtigen wollen oder nicht. Dies kann einfach durch die Spezifikation der entsprechenden Kerntechnologiekategorien mit Hilfe des Schlüsselwortes `CORE` in der verwendeten Graphbeschreibung definiert werden.

Darüber hinaus haben alle Experten (100%) die präsentierten Restrukturierungsempfehlungen als sehr hilfreich empfunden. Nach ihrer Einschätzung sind solche Restrukturierungsempfehlungen nützlich, um echte Potentiale für Restrukturierungen zu identifizieren. Fünf Experten (83%) gaben an, dass die vorgestellten Empfehlungen dazu beitragen, objektiv nachvollziehbare Restrukturierungsentscheidungen zur Verringerung der Variabilität treffen zu können. Jedoch merkten auch vier Architekten (66%) an, dass die Nützlichkeit der Restrukturierungsempfehlungen stark von den Geschäftsregeln und ihren individuellen Kriterien und Schwellwerten abhängt. Dies wurde bereits auch schon in der Fallstudie B ersichtlich. Daher ist unser Ansatz darauf ausgelegt, unternehmensspezifische Anforderungen und Ziele in den Geschäftsregeln berücksichtigen zu können. Somit ist es Experten möglich, die anzuwendenden Geschäftsregeln so zu spezifizieren, dass geeignete Restrukturierungsempfehlungen auf Basis ihrer individuellen Anforderungen generiert werden können.

Bewertung und Restrukturierungsentscheidung

FF-I.1 – Verständlichkeit: Zuerst haben wir den Experten die Ergebnisse der Differenzanalyse für die betrachteten Restrukturierungsempfehlungen zur Bewertung vorgelegt. Alle Experten (100%) haben ausgesagt, dass sie die Idee des Ansatzes gut und verständlich finden. Fünf von Ihnen (83%) haben auch die Verständlichkeit der unterschiedlichen Differenzwerte bestätigt. Lediglich ein Experte (17%) sagte aus, dass diese Werte zu feingranular wären und daher schwierig zu verstehen sind. Auf der anderen Seite haben zwei Experten (33%) besonders hervorgehoben, dass die detaillierten Differenzwerte für unterschiedliche Variabilitätsbeziehungen gut geeignet sind, um das Reduzierungspotential im Detail zu inspizieren. Da unser Ansatz sowohl detaillierte Differenzwerte als auch einen aggregierten Indikator für die Variabilitätsdifferenz (vgl. Gleichung 5.1) liefert, können die Anforderungen aller Experten berücksichtigt werden.

Anschließend haben wir den Experten auch die Ergebnisse der Abhängigkeitsanalyse und der Aufwandsschätzung für die ausgewählten Restrukturierungsempfehlungen präsentiert. Hierbei waren alle Experten (100%) der Meinung, dass diese Ansätze und ihre entsprechenden Ergebnisse gut verständlich und einfach anwendbar sind. Vier Experten (66%) sagten zudem aus, dass die Unterscheidung zwischen bekannten und unbekannten Abhängigkeiten sowie ihre unterschiedliche Berücksichtigung im Rahmen der Aufwandsschätzung sinnvoll ist.

FF-I.2 – Nützlichkeit: Hinsichtlich der präsentierten Differenzwerte sagten fünf Experten (83%) aus, dass die resultierenden Indikatoren für die unterschiedlichen Variabilitätsrelationen (z.B. alternativ und optional) sehr hilfreich sind, da sie den Architekten einen schnellen Überblick über das mögliche Reduzierungspotential einer konkreten Restrukturierungsempfehlung liefern.

In Bezug auf die identifizierten Abhängigkeiten erwähnten alle Experten (100%), dass die Informationen über konkrete Abhängigkeiten einer Restrukturierungsempfehlung sehr wertvoll sind, da es für sie selbst unmöglich ist, solche Abhängigkeiten für eine große Anzahl von TAPs manuell zu ermitteln. Drei der Experten (50%) haben zudem erklärt, dass es mit diesen Abhängigkeiten möglich ist, auf die wesentlichen Aspekte einer Restrukturierung zu fokussieren. Denn die identifizierten Abhängigkeiten stellen die technischen Herausforderungen einer konkreten Restrukturierung heraus, welche es für eine erfolgreiche Umsetzung zu lösen gilt. Auf der anderen Seite waren alle Experten (100%) der Meinung, dass der entwickelte Ansatz auch die Fähigkeit besitzen sollte, die Ergebnisse einer manuellen Überprüfung von konkreten Abhängigkeiten so zu speichern, dass dieses von Experten ergänzte Domänenwissen für nachfolgende Analysen verfügbar gemacht werden kann. Dies hätte den Vorteil, dass hinzugefügtes Expertenwissen wiederverwendet und somit der Aufwand für die manuelle Überprüfung gleicher Abhängigkeiten reduziert werden kann. Da hierfür allerdings eine umfangreiche Weiterentwicklung unseres Ansatzes hin zu einem lernenden System erforderlich ist, kann dies nicht mehr im Rahmen dieser Arbeit berücksichtigt werden. Es sollte aber als eine vielversprechende Erweiterung unseres Ansatzes in zukünftigen Arbeiten betrachtet werden. Somit ist unser Ansatz momentan darauf beschränkt, dass manuelle Überprüfungen für jede Restrukturierungsempfehlung einzeln durchgeführt werden müssen. Allerdings ist der damit verbundene Aufwand lediglich leicht erhöht, da die manuelle Überprüfung von Abhängigkeiten nur für solche Restrukturierungsempfehlungen erforderlich ist, die für eine Umsetzung von Experten auch in Betracht gezogen werden. Insofern waren alle interviewten Architekten (100%) dennoch von der Nützlichkeit der automatisiert ermittelten Abhängigkeiten überzeugt.

Bei der Beurteilung der präsentierten Aufwände haben fünf Experten (83%) beschrieben, dass die Migrations- und Anpassungsaufwände hilfreich sind, um wertvolle Restrukturierungsempfehlungen identifizieren zu können. Drei Experten (50%) haben dabei erwähnt, dass sie es besonders nützlich finden, dass sich der Anpassungsaufwand an den anpassungsabhängigen Komponenten und deren Typ (bekannt oder unbekannt) orientiert. Allerdings hat ein weiterer Experte (16%) angemerkt, dass die differenzierte Betrachtung der Aufwände für diese unterschiedlichen Abhängigkeitstypen auch zu einer falschen Aufwandsschätzung führen kann. Dies begründet er damit, dass zwar eventuell fehlendes Technologiewissen berücksichtigt wird, aber aus fehlendem Wissen nicht immer auch höherer Anpassungsaufwand resultieren muss. Diese Einschätzung kann in Einzelfällen zutreffend sein. Unsere Fallstudien haben gezeigt, dass die ermittelten Anpassungsaufwände im Durchschnitt zu 87% mit den manuellen Schätzungen übereinstimmen. Somit ist der Grad an korrekt eingeschätzten Anpassungsaufwänden zwar sehr hoch, jedoch nicht zu 100% erreicht. Daraus

lässt sich schließen, dass unser Ansatz Experten bei der Aufwandsschätzung umfangreich unterstützt, jedoch nicht jeden Einzelfall im Detail berücksichtigen kann. Vor der Umsetzung konkreter Restrukturierungsempfehlungen sollten Experten daher nochmals alle Bewertungsdetails genau prüfen. Jedoch ist es erst durch unseren Ansatz möglich, dass sich Experten auf wenige, vielversprechende Restrukturierungsmöglichkeiten in einer gewachsenen IT-Landschaft fokussieren können. Ein weiterer Experte (16%) hat ausgesagt, dass eine Aufwandsmatrix mit feingranulareren Werten auf Ebene der Komponentenkategorien hilfreich für noch präzisere Aufwandsschätzungen sein kann. Dies wäre mit unserem Ansatz auch realisierbar, indem in einer Aufwandsmatrix neben dem Layer und Tier auch die jeweilige Kategorie spezifiziert werden würde. Hierfür müsste lediglich die entsprechende DSL um Kategorien erweitert werden. Da der manuelle Aufwand zur Spezifizierung der Aufwandsmatrix damit jedoch steigt und nur einer der sechs befragten Experten diese Erweiterung für sinnvoll erachtet hat, haben wir dies im Rahmen unserer Arbeit nicht mehr berücksichtigt. Eine entsprechende Erweiterung unseres Ansatzes kann aber auch in zukünftigen Arbeiten betrachtet werden. Da wir in unseren Fallstudien eine hohe Übereinstimmung zwischen automatisiert bestimmten und manuell verifizierten Aufwänden beobachten konnten, sind wir dennoch überzeugt, dass unsere Aufwandsschätzung gut geeignet ist, um Experten zu unterstützen.

Abschließend haben wir den Experten noch die Ergebnisse der Portfolioanalyse für das betrachtete Set von TAPs präsentiert und die Simulation exemplarisch für einige Restrukturierungsempfehlungen durchgeführt. Alle Experten (100%) waren sich einig, dass die Simulation von ausgewählten Restrukturierungsempfehlungen und die Visualisierung der resultierenden Soll-Architektur einen enorm hohen Mehrwert bieten. Vier Experten (66%) haben auch hervorgehoben, dass unser iteratives Vorgehen mit der Simulation aufeinander aufbauender 150%-Modelle in Verbindung mit der Möglichkeit des Rückschrittes zu einem vorherigen 150%-Modell besonders hilfreich ist, um geeignete Restrukturierungsentscheidungen treffen zu können. Zudem haben alle Architekten (100%) auch die Ergebnisse der Portfoliobewertung für nützlich erachtet, da diese dabei unterstützen, die erzeugten Restrukturierungsempfehlungen gegeneinander abzuwägen und solche zu identifizieren, die besonders lohnenswert sind und für eine Umsetzung in Betracht gezogen werden sollten.

Zusammenfassung aller drei Bereiche

FF-I.1 – Verständlichkeit: Zusammenfassend lässt sich feststellen, dass unser Ansatz in allen drei analysierten Bereichen verständlich und gut anwendbar ist. So hat sich gezeigt, dass fast alle Experten das grundlegende Verständnis für die vorgeschlagenen Techniken hatten und die gezeigten Artefakte (z.B. Geschäftsregeln) richtig interpretieren konnten. Nur vereinzelt traten kleinere Verständnisschwierigkeiten auf (z.B. der `GIVEN`-Teil einer Geschäftsregel), welche aber durch eine bessere Einführung der Experten und durch zusätzliche Beispiele beseitigt werden können. Insofern kann die Forschungsfrage *FF-I.1* insgesamt positiv beantwortet werden.

FF-I.2 – Nützlichkeit: Die interviewten Experten haben die Nützlichkeit der durch unseren Ansatz generierten Ergebnisse als sehr hoch eingeschätzt. So waren alle Experten überzeugt, dass ein konkretes 150%-Modell sehr hilfreich ist, um schnell und einfach alle Gemeinsamkeiten und Unterschiede in den analysierten TAPs zu erkennen. Zudem waren sich die meisten Experten auch einig, dass die abgeleiteten Restrukturierungsempfehlung und ihre Bewertungsergebnisse nützlich sind, um lohnenswerte Restrukturierungen zu identifizieren. Und auch die Simulation im Rahmen unseres iterativen Vorgehens haben die Experten als sehr hilfreich eingeschätzt, da sie hiermit geeignete und objektiv nachvollziehbare Entscheidungen zur Restrukturierung von betrachteten TAPs treffen können. Zwar wurden auch Potentiale für zukünftige Verbesserungen identifiziert (z.B. Weiterentwicklung zu einem lernenden System, feingranularere Aufwandsmatrix), dennoch konnten die generierten Ergebnisse unseres Ansatz die befragten Experten hinsichtlich ihrer Nützlichkeit überzeugen. Somit lässt sich auch die Forschungsfrage *FF-I.2* positiv beantworten.

6.4 Validität der Evaluationsergebnisse

Obwohl wir unsere Ansätze sorgfältig entworfen, implementiert und evaluiert haben, können bestimmte Umstände Auswirkungen auf die Validität der Ergebnisse unserer Fallstudien und der durchgeführten Experteninterviews haben. Diese werden nachfolgend kurz beschrieben.

Anzahl der Experten: Die Evaluationsergebnisse basieren auf dem Wissen und der Bewertung von insgesamt sechs Experten unseres Industriepartners. Obwohl all diese Experten über eine mehrjährige Erfahrung als professioneller IT-Architekt im industriellen Kontext verfügen, könnte die Einbeziehung anderer Experten zu abweichenden Ergebnissen in den Interviews und den Fallstudien führen. Allerdings erfolgte die Auswahl der Experten so, dass eine möglichst breite Streuung in Bezug auf ihre Berufserfahrung und ihren Tätigkeitsschwerpunkt realisiert wurde. Wir sind daher überzeugt, dass die Expertenbewertungen, auch mit Blick auf die sehr positiven Interviewergebnisse, als valide betrachtet werden können.

Subjektive Beurteilung der Experten: Einige Bestandteile unseres Ansatzes zur Variabilitätsreduzierung bauen auf subjektiven Beurteilungen unserer Experten auf. Dies trifft auf die zu erstellenden Artefakte, wie beispielsweise auf die spezifizierten Geschäftsregeln, das entwickelte Abhängigkeitsmodell und die definierte Aufwandsmatrix zu. Durch unterschiedliche Spezifikationen dieser Artefakte können die Resultate unseres automatisierten Ansatzes allerdings variieren, wodurch auch abweichende Ergebnisse in den Fallstudien möglich sind. Jedoch ist unser Ansatz bewusst so konzipiert, dass Geschäftsregeln, Abhängigkeitsmodelle und andere Artefakte den individuellen Anforderungen der jeweiligen Unternehmen angepasst werden können. Nur so ist eine Berücksichtigung der unternehmensspezifischen Anforderungen und Ziele im Rahmen der Variabilitätsreduzierung möglich.

Voreingenommenheit der Experten: Vor Durchführung der Fallstudien haben wir zusammen mit drei der Experten, die uns zur Verfügung standen, die erforderlichen Artefakte für unseren Ansatz (z.B. den Geschäftsregelkatalog und die Aufwandsmatrix) entwickelt und spezifiziert. Hierbei haben die Experten konstruktiven Input in die Ausarbeitung der Methodik und der resultierenden Artefakten gegeben. Dieselben Experten waren anschließend aber auch an den Fallstudien und den Interviews beteiligt. Daher lässt sich nicht ausschließen, dass drei der sechs verfügbaren Experten voreingenommen waren und in gewisser Weise ihre eigene Arbeit mit bewertet haben. Nichtsdestotrotz können wir feststellen, dass die Ergebnisse aus den Fallstudien und den Experteninterviews insgesamt sehr positiv ausgefallen sind und sich dabei die Bewertungen der drei an der Vorarbeit beteiligten Experten nicht maßgeblich von den Bewertungen der anderen Experten unterschieden haben. Somit können keine Auswirkungen einer Voreingenommenheit festgestellt werden, allerdings lässt sich diese auch nicht ganz ausschließen.

Eingeschränkte manuelle Detailanalyse: Im Rahmen der Fallstudien haben wir unsere Experten gebeten, verschiedene Ergebnisartefakte, welche automatisiert generiert wurden (z.B. 150%-Modelle und Restrukturierungsempfehlungen), manuell zu überprüfen. Allerdings waren die Experten oftmals zeitlich nur sehr eingeschränkt verfügbar, sodass die Evaluierung aller automatisiert erstellten Ergebnisse nicht möglich war und nur eine begrenzte manuelle Detailanalyse durchgeführt wurde. Jedoch haben unsere Experten hierfür eine zufällige Auswahl der zu untersuchenden Artefakte getroffen und diese dann im Detail analysiert. Darüber hinaus wurden auch andere Ergebnisartefakte stichprobenartig überprüft, woraus sich ein Gesamtbild für die Experten ergeben hat. Insofern können wir davon ausgehen, dass auch die eingeschränkte manuelle Detailanalyse zu validen Bewertungsergebnissen geführt hat.

Auswahl der untersuchten TAPs: Die Ergebnisse unserer Fallstudien basieren auf vier unterschiedlichen Sets von TAPs, die ausgewählte Softwaresysteme repräsentieren. Während die vorgestellten Evaluationsergebnisse für diese Softwaresysteme valide sind, könnte die Analyse von TAPs anderer Softwaresysteme zu abweichenden Ergebnissen führen. Daher haben wir die analysierten Softwaresysteme sorgfältig in Zusammenarbeit mit unseren Experten ausgewählt. Hierbei wurden solche TAPs berücksichtigt, die eine wichtige Kerntechnologie unseres Industriepartners beinhalten und über eine signifikante Anzahl von implementierten Komponenten verfügen. So konnten wir Sets mit passenden Stichproben für eine aussagekräftige Evaluierung bilden. Daher sind wir überzeugt, dass die technologisch unterschiedlichen Sets und ihre Größe mit bis zu 12.405 Komponenten geeignet sind, um die Anwendbarkeit unseres Ansatzes zur Variabilitätsreduzierung im Rahmen der vorgestellten Fallstudien zu zeigen.

Verfügbarkeit der erforderlichen Daten: Unser Ansatz benötigt bestimmte Daten über technische Architekturen von Softwaresystemen sowie zusätzliche Informationen zu den spezifischen Eigenschaften von implementierten Komponenten (z.B. Strategischer Fit und Kosten), um effektiv zu funktionieren. Solche Informationen sind aber nicht immer in allen Unternehmen vorhanden. Und auch bei unserem Industriepartner konnten wir nicht alle erforderlichen Daten für die Fallstudien erhalten (z.B. Funktionen von Komponenten). Dies schränkt die Genauigkeit der Analyseergebnisse unseres Ansatzes ein, sodass beispielsweise Ersetzungsempfehlungen für solche Komponenten generiert wurden, die funktional nicht deckungsgleich sind. Wir haben allerdings gezeigt, dass unser vorgeschlagener Ansatz nur wenige Daten als notwendigen Input benötigt, um trotzdem hilfreiche Ergebnisse zur Unterstützung von Domänenexperten für die Variabilitätsreduzierung zu generieren. Insofern kann unser Ansatz bereits mit einfachen Informationen über die technische Architektur von Softwaresystemen verwendet werden. Je mehr spezifische Daten jedoch für die automatisierten Analysen zur Verfügung gestellt werden können, desto präziser und hilfreicher sind deren Ergebnisse.

6.5 Zusammenfassung

In diesem Kapitel wurde die Evaluation unseres entwickelten Ansatzes zur Reduzierung der Variabilität von Technologiearchitekturen in gewachsenen IT-Landschaften (vgl. Kapitel 3 bis 5) vorgestellt. Zu diesem Zweck haben wir drei Fallstudien mit realen Architekturdaten und Interviews mit verschiedenen Experten unseres Industriepartners durchgeführt.

In der *Fallstudie A* haben wir die Eignung unseres entworfenen Ansatzes zur Bestimmung der Variabilität in Technologiearchitekturen evaluiert. Hierbei konnten wir zeigen, dass unser Ansatz in der Lage ist, explizite Variabilitätsbeziehungen für eine beliebige Anzahl an TAPs automatisiert zu bestimmen. Zudem bekräftigen die Ergebnisse dieser Fallstudie, dass der vorgeschlagene Ansatz gut skalierbar ist und dabei eine angemessene Laufzeit bietet.

In der *Fallstudie B* wurde anschließend evaluiert, ob der entwickelte Ansatz zur automatischen Ableitung von konkreten Restrukturierungsempfehlungen geeignet ist. Hier konnten wir zeigen, dass unser Ansatz in der Lage ist, spezifische Restrukturierungspotentiale für unterschiedliche Sets von TAPs abzuleiten und dass die daraus generierten Empfehlungen zur Reduzierung unnötiger Variabilität beitragen können.

Im Rahmen der *Fallstudie C* wurde eine Evaluierung der automatisierten Methoden zur Bewertung und Simulation von Restrukturierungsempfehlungen durchgeführt. Hier konnten wir zeigen, dass die vorgeschlagenen Techniken geeignet sind, um einzelne Restrukturierungsempfehlung sowie das gesamte Portfolio aller Empfehlungen zu bewerten und eine entsprechende Soll-Architektur zu simulieren.

In den *Interviews* wurde bestätigt, dass die generierten Ergebnisse unseres Ansatzes gut verständlich und hilfreich für Experten sind. So bieten sie unter anderem eine geeignete Grundlage für objektiv nachvollziehbare Restrukturierungsentscheidungen.

7 Zusammenfassung der Arbeit

In diesem Kapitel wird eine Zusammenfassung der vorliegenden Doktorarbeit gegeben. Diese beinhaltet eine kurze Beschreibung der wissenschaftlichen Beiträge im Abschnitt 7.1, eine Diskussion der getroffenen Design-Entscheidungen für die vorgeschlagenen Ansätze im Abschnitt 7.2 sowie einen Ausblick über weitergehende Forschungsthemen für den Kontext dieser Arbeit im Abschnitt 7.3.

7.1 Wissenschaftliche Beiträge

Im Rahmen dieser Arbeit wurde untersucht, wie Domänenexperten bei der Reduzierung unnötiger Variabilität von Technologiearchitekturen in großen, gewachsenen IT-Landschaften unterstützt werden können. Zu diesem Zweck wurden drei Forschungsfragen aufgestellt und beantwortet.

FF1 – *Wie können explizite Variabilitätsbeziehungen zwischen individuellen Technologiearchitekturen identifiziert werden?*

Mit Hilfe unseres entwickelten Mining-Verfahrens (vgl. Kapitel 3) konnten wir zeigen, dass die Bestimmung der Variabilität für Technologiearchitekturen unterschiedlichster Softwaresysteme automatisiert durchführbar ist und hierdurch inhärente Variabilitätsbeziehungen identifiziert werden können. Der beschriebene Ansatz ist dabei in der Lage, eine beliebige Anzahl von systemspezifischen Technologiearchitekturen, auch als *Technologiearchitekturplan (TAP)* bezeichnet, gleichzeitig zu analysieren und deren Gemeinsamkeiten und Unterschiede zu bestimmen. Diese Komponenten und ihre Variabilitätsinformationen werden anschließend in einem einzigen Modell, dem sogenannten *150%-Modell*, abgebildet. Zu diesem Zweck werden zuerst alle technologischen Komponenten der gleichen Layer-Tier-Intersection eines jeweiligen TAPs in einem Cluster zusammengefasst. Anschließend werden einzelnen Konfigurationsoptionen durch geeignete Splitting-Regeln extrahiert und entsprechende Feincluster gebildet. So können in einem dritten Schritt alle konkreten Variabilitätsbeziehungen zwischen den einzelnen Komponenten der jeweiligen Feincluster identifiziert und diese Komponenten dann mit ihren jeweiligen Variabilitätsinformationen in ein gemeinsames 150%-Modell zusammengeführt werden. Im Rahmen einer Fallstudie konnten wir feststellen, dass die generierten 150%-Modelle für zuvor ausgewählte TAPs korrekt und vollständig waren. Zudem hat sich bei einer weiteren Untersuchung mit Hilfe von Experteninterviews gezeigt, dass der vorgeschlagene Ansatz verständlich und sehr nützlich ist. So haben die befragten Domänenexperten herausgestellt, dass die Variabilitätsbeziehungen zwischen technologischen Komponenten nun schnell und ohne

großen manuellen Aufwand identifiziert werden können, was zu einem hohen Mehrwert für die tägliche Architekturarbeit führt. Zudem haben die Experten auch betont, dass solch ein 150%-Modell eine gute Übersicht über die eingesetzten Technologien bietet und dadurch bereits eine manuelle Identifizierung von Restrukturierungspotentialen ermöglicht wird.

FF2 – *Wie können konkrete Restrukturierungsempfehlungen zur Reduzierung unnötiger Variabilität bestimmt werden?*

Zur Beantwortung dieser Forschungsfrage haben wir einen regelbasierten Ansatz zur automatischen Ableitung von konkreten Restrukturierungsempfehlungen entwickelt (vgl. Kapitel 4). Hierbei steht die Reduzierung der unnötigen Variabilität in den betrachteten TAPs im Vordergrund. Zu diesem Zweck wird zuerst eine Businessanalyse durchgeführt, welche durch die Verwendung von spezifizierten Geschäftsregeln unnötig variable Komponenten identifizieren und geeignete Restrukturierungspotentiale zu deren Reduzierung ermitteln kann. In einem zweiten Schritt wird dann die technische Machbarkeit dieser Restrukturierungspotentiale analysiert, indem die Ähnlichkeit der *variantenspezifischen Komponentengraphen (VKG)* der betroffenen Komponenten festgestellt und eventuell vorhandene technische Abhängigkeiten bestimmt werden. Unter Berücksichtigung der ermittelten Restrukturierungspotentiale und deren technischer Machbarkeit können anschließend in einem dritten Schritt konkrete Restrukturierungsempfehlungen abgeleitet werden, die zur Reduzierung der unnötigen Variabilität in den betrachteten TAPs beitragen können. Mit Hilfe einer Fallstudie konnten wir zeigen, dass der vorgeschlagene Ansatz in der Lage ist, geeignete Restrukturierungsempfehlungen für unterschiedliche Sets von ausgewählten TAPs abzuleiten. Zudem hat sich durch die Experteninterviews bestätigt, dass unser Ansatz verständlich ist und seine Ergebnisse nützlich für Domänenexperten sind. So hoben die befragten Experten beispielsweise hervor, dass sie durch die generierten Empfehlungen echte Restrukturierungsmöglichkeiten aufgezeigt bekommen und entsprechende Restrukturierungsentscheidungen damit nachvollziehbar sind und besser begründet werden können.

FF3 – *Wie können Domänenexperten bei der Entscheidungsfindung zur Umsetzung von Restrukturierungsempfehlungen unterstützt werden?*

Zur Unterstützung von Domänenexperten bei der Entscheidungsfindung haben wir einen Ansatz entwickelt, der sich zur automatischen Bewertung und Simulation von abgeleiteten Restrukturierungsempfehlungen eignet und so Experten bei der Findung von konkreten Restrukturierungsentscheidungen für die analysierten TAPs unterstützt (vgl. Kapitel 5). Zu diesem Zweck haben wir Methoden zur Bewertung von einzelnen Restrukturierungsempfehlungen sowie zur Bewertung des Gesamtportfolios aller Empfehlungen vorgestellt. Die Ergebnisse, beispielsweise die potentiellen Aufwände einer Empfehlung oder die Aufwand-Nutzen-Matrix des Gesamtportfolios, bieten Domänenexperten eine geeignete quantitative Entscheidungsgrundlage, um

über konkrete Restrukturierungsmaßnahmen zur Reduzierung der Variabilität in den betrachteten TAPs entscheiden zu können. Zur weiteren Unterstützung können Domänenexperten die für sie interessanten Restrukturierungsempfehlungen auswählen, um diese erst einmal zu simulieren. Hierfür generiert unser Ansatz das resultierende Soll-150%-Modell, welches die entstehende Soll-Architektur mit ihrer entsprechenden Variabilität repräsentiert. Auf dieser Grundlage können Experten dann entscheiden, ob sie die ausgewählten Empfehlungen umsetzen oder andere Empfehlungen zur Simulation auswählen wollen. Auch eine Ableitung neuer Restrukturierungsempfehlungen für das Soll-150%-Modell ist mit unserem iterativen Vorgehen möglich, sodass sich Domänenexperten schrittweise an ihre optimale Architekturlösung herantasten können. Im Rahmen einer Fallstudie konnten wir zeigen, dass die vorgeschlagenen Methoden zur automatischen Bewertung und Simulation von abgeleiteten Restrukturierungsempfehlungen geeignet sind, um Domänenexperten zu unterstützen. Außerdem ließ sich mit Hilfe durchgeführter Interviews feststellen, dass dieser Ansatz von den Experten verstanden wird und seine Ergebnisse als sehr nützlich betrachtet werden. Beispielsweise waren alle befragten Experten überzeugt, dass sie mit den Bewertungsergebnissen für Restrukturierungsempfehlungen eine geeignete Grundlage haben, um objektiv nachvollziehbare Entscheidungen treffen zu können. Demnach helfen die einzelnen Bewertungsergebnisse, die jeweiligen Restrukturierungsempfehlungen gegeneinander abzuwägen und lohnenswerte Restrukturierungen zu identifizieren. Darüber hinaus haben die Experten auch die Möglichkeit zur Simulation von Restrukturierungsmaßnahmen besonders hervorgehoben. Hierdurch ist es ihnen erstmals möglich, schon frühzeitig über das Zielbild einer Restrukturierung zu diskutieren und Entscheidungen anhand von zukünftigen Entwicklungen festzumachen.

Die beschrieben Ansätze zur Beantwortung der drei definierten Forschungsfragen wurden in einen durchgängigen Gesamtansatz integriert. Mit Hilfe dieses Gesamtansatzes lässt sich auch die zentrale Forschungsfrage dieser Doktorarbeit beantworten:

Wie können Domänenexperten bei der Reduzierung unnötiger Variabilität von Technologiearchitekturen in großen, gewachsenen IT-Landschaften unterstützt werden?

Der hierfür entwickelte Gesamtansatz besteht aus sieben Phasen und ist in Abbildung 7.1 dargestellt. In *Phase A* werden die zu untersuchenden TAPs durch Domänenexperten manuell für die Analyse ausgewählt. In *Phase B* erfolgt eine semiautomatische Aufbereitung dieser TAPs zur Reduzierung eventuell vorhandener Datenfehler. In der *Phase C* wird dann die Variabilitätsbestimmung automatisiert durchgeführt und das entsprechende Ist-150%-Modell erstellt. Danach erfolgt in *Phase D* die automatische Ableitung konkreter Restrukturierungsempfehlungen zur Reduzierung der unnötigen Variabilität. Anschließend wird in der *Phase E* jede einzelne dieser Restrukturierungsempfehlungen automatisch bewertet. In *Phase F* wird dann eine automatisierte Bewertung des Gesamtportfolios aller Restrukturierungsempfehlungen durchgeführt. Alle Bewertungsergebnisse dienen so als quantitative Grundlage

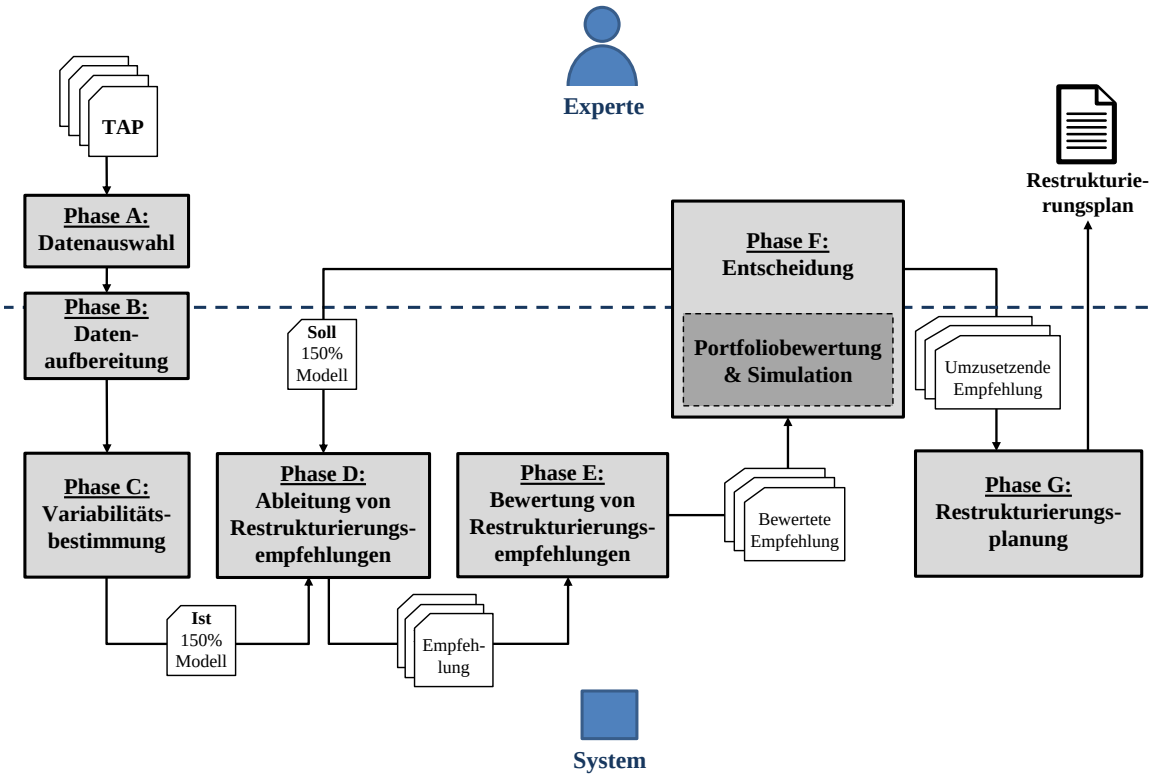


Abbildung 7.1: Workflow für den Gesamtansatz

für die manuelle Entscheidungsfindung der Domänenexperten. Unterstützt werden diese auch durch eine automatisierte Methode zur Simulation beliebiger Restrukturierungsempfehlungen, wodurch ein Soll-150%-Modell entsteht. Für dieses können weitere Empfehlungen abgeleitet werden, indem dieses Soll-Modell als erneuter Input für die Phase D verwendet wird. So ist ein iteratives Vorgehen bis zum Erreichen einer gewünschten Architekturlösung möglich. Haben sich Experten dann für die Umsetzung bestimmter Empfehlungen entschieden, wird automatisch ein Restrukturierungsplan mit den erforderlichen Migrations- und Anpassungsmaßnahmen für die beteiligten Systeme in *Phase G* erzeugt und den Domänenexperten für die Initiierung der entsprechenden Veränderungsprojekte zur Verfügung gestellt.

7.2 Diskussion

Im Laufe der Entwicklung unseres Ansatzes haben wir einige Design-Entscheidungen getroffen, welche Auswirkungen auf das vorgeschlagene Verfahren haben. Diese werden im Folgenden kurz erläutert und diskutiert:

Definition von Schwellwerten: Die Funktionsweise unseres Ansatzes basiert teilweise auf frei definierbaren Schwellwerten. So werden beispielsweise Schwellwerte für die Überprüfung von Kriterien einer Geschäftsregel oder zur Bestimmung der

Ähnlichkeit von zwei variantenspezifischen Komponentengraphen (VKG) verwendet. Anhand von Gesprächen mit Domänenexperten unseres Industriepartners und der Auswertung von verfügbaren Industriedaten haben wir die jeweiligen Schwellwerte für unseren Ansatz diskutiert und nach bestem Wissen definiert. Die Ergebnisse unseres Verfahrens sind allerdings von diesen Schwellwerten abhängig, sodass eine Änderung an den Schwellwerten auch eine andere Ergebnismenge zufolge haben kann. Aus diesem Grund haben wir unseren Ansatz so entworfen, dass die Schwellwerte frei konfigurierbar sind. So ist es Experten möglich, die vorgeschlagenen Werte an unternehmensindividuelle Bedingungen anzupassen. So kann zum Beispiel der Schwellwert H=10% in der Geschäftsregel `GR2` (vgl. Listing 4.6) zur Eliminierung von optionalen Konfigurationsoptionen höher oder niedriger gesetzt werden.

Festlegung der Reihenfolge der Phasen und ihrer einzelnen Schritte: Einen weiteren Einfluss auf die Ergebnisse unseres Ansatzes hat die Reihenfolge der entworfenen Phasen und ihrer einzelnen Schritte. So wird beispielsweise zur Ableitung von Restrukturierungsempfehlungen für die identifizierten Kandidaten für unnötige Variabilität zuerst eine businessorientierte und anschließend eine technologieorientierte Analyse durchgeführt. Wir haben dies im Laufe unserer Arbeit mit Domänenexperten diskutiert und sind zu dem Entschluss gekommen, dass eine Beurteilung von variablen Komponenten zuerst aus einer fachlichen Perspektive erfolgen muss. Denn ein Unternehmen kann hier unterschiedliche Technologiestrategien verfolgen (z.B. kostengünstige IT-Infrastruktur ohne fachliche Redundanzen versus Multi-Lieferanten-Strategie mit fachlichen Redundanzen), was sich in den Geschäftsregeln für die Businessanalyse (vgl. Abschnitt 4.1) niederschlägt. Würde die graphbasierte Technologieanalyse zuerst durchgeführt werden, könnten technische Restrukturierungspotentiale identifiziert werden, die aus fachlicher Sicht aber nicht gewollt sind. Da der Gesamtansatz letztlich dazu beitragen soll, die Unternehmensziele besser zu erreichen, ist demnach die regelbasierte Businessanalyse zuerst durchzuführen. So können Restrukturierungsempfehlungen für eine Vielzahl von Kandidaten effizienter abgeleitet werden.

Darüber hinaus kam in Gesprächen mit Experten auch die Frage auf, ob Abhängigkeiten im Rahmen der Bewertung nicht zuerst ermittelt werden sollten, um Inkompatibilitäten oder notwendige Anpassungen, die einer Restrukturierung entgegen stehen würden, frühzeitig zu berücksichtigen. Da unser Ansatz aber ohne Informationen über voneinander abhängige Elemente nur Potentiale für solche Abhängigkeiten automatisiert identifizieren kann, sind diese ohnehin durch Domänenexperten zu verifizieren, um daraus weitere Schritte ableiten zu können. Zwar konnten wir im Rahmen der Fallstudie C zeigen, dass die automatisch ermittelten Abhängigkeiten im Durchschnitt zu über 80% korrekt waren, jedoch reicht dies nicht aus, um einzelne Restrukturierungsempfehlungen automatisch zu löschen, falls entsprechende Abhängigkeiten erkannt wurden. Dies kann nur durch Experten mit ihrem Domänenwissen erfolgen. Da unser Ansatz allerdings darauf ausgelegt ist, Experten bei dieser Entscheidung zu unterstützen, werden ihnen sämtliche Ergebnisse aus der Bewertung von Restrukturierungsempfehlungen zur Verfügung gestellt. Insofern hat es keinen

Einfluss auf die Ergebnisse unseres Ansatzes, wann die Abhängigkeitsanalyse durchgeführt wird. Allerdings muss diese Analyse erfolgen, bevor der potentielle Aufwand einer Restrukturierungsempfehlung geschätzt wird, da hierfür die anpassungsabhängigen Elemente bekannt sein müssen.

Fokussierung auf logische Elemente: Im Rahmen der Entwicklung unseres Ansatzes hat sich schnell die Frage aufgetan, ob dieser Ansatz auf Ebene der logischen Elemente (abstrakte Komponente, z.B. Windows) oder auf Ebene der physischen Elemente (konkrete Komponente (einsetzbare Version), z.B. Windows 10 Professional) unterstützen soll. Diese Frage haben wir mit den Domänenexperten unseres Industriepartners diskutiert und gemeinsam festgelegt, dass zwar die Variabilitätsanalyse für beide Ebenen ermöglicht werden soll, aber Restrukturierungsempfehlungen lediglich auf Ebene der logischen Elemente notwendig sind. Begründet haben die Experten ihre Einschätzung damit, dass die einzig mögliche Restrukturierungsempfehlung auf Ebene der physischen Elemente beinhalten kann, dass eine ältere Komponenten-Version gegen eine neue ersetzt werden soll. Hierfür ist ihrer Meinung nach aber kein automatisierter Ansatz notwendig, da die Experten diesen Rückschluss sehr einfach manuell aus dem 150%-Modell ziehen können. Aus diesem Grund fokussiert unser Ansatz hauptsächlich auf die logischen Elemente in TAPs und berücksichtigt physische Elemente nur im Rahmen der Variabilitätsanalyse. Jedoch könnten auch die anderen Methoden leicht für physische Elemente erweitert werden, indem diese die entsprechenden Modellelemente aus dem generierten 150%-Modell weiterverarbeiten.

7.3 Ausblick

Die in dieser Arbeit vorgestellten Ansätze können im Rahmen zukünftiger Arbeiten noch erweitert werden. Im Laufe der Entwicklung und Evaluation des präsentierten Gesamtansatzes sind hierzu weitere potentielle Forschungsthemen diskutiert worden:

Unterstützung durch eine automatisierte Auswahl von TAPs: Unser Ansatz sieht in der *Phase A* vor, dass Experten ihr Domänenwissen nutzen, um technisch verwandte TAPs für die Analyse und die spätere Restrukturierung manuell auszuwählen. Dies bedeutet allerdings, dass diese Phase den größten manuellen Aufwand im Gesamtansatz erforderlich macht. Mit Hilfe von automatisierten Methoden ließe sich dieser manuelle Aufwand noch weiter reduzieren. Zu diesem Zweck kann ein Verfahren entwickelt werden, welches in der Lage ist, eine beliebige Menge von TAPs anhand ihrer technischen Merkmale zu gruppieren. Hierzu eignen sich Clustering-Techniken aus dem Bereich des *Data Minings*. Mit diesen ist es beispielsweise möglich, Cluster von ähnlichen Objekten automatisiert zu bestimmen [Ber06, PST12]. Damit ließen sich auch Cluster von technisch ähnlichen TAPs automatisch erzeugen, welche dann für die weitere Analyse im Rahmen des gezeigten Ansatzes verwendet werden können.

Weiterentwicklung hin zu einem lernenden System: Während der Evaluation sowie im Rahmen von Diskussionen mit Experten hat sich herausgestellt, dass unserer Ansatz davon profitieren würde, wenn dieser in der Lage wäre, einmalig durch Experten hinein gegebenes Domänenwissen zu speichern und für nachfolgende Analysen verfügbar zu machen. Dies haben Experten vor allem im Kontext der Entscheidungsfindung betont. Hier wäre es ihrer Meinung nach wünschenswert, wenn die Ergebnisse einer manuellen Verifizierung von Abhängigkeiten einer konkreten Restrukturierungsempfehlung auch für andere Restrukturierungsempfehlungen nutzbar wären. Zudem haben sie vorgeschlagen, dass auch ihre Bewertung einer Restrukturierungsempfehlung für die nachfolgenden Ableitungen neuer Empfehlungen Berücksichtigung finden könnte. So wäre es beispielsweise denkbar, dass eine durch Experten als *ungeeignet* bewertete Restrukturierungsempfehlung in einer nachfolgenden Analyse nicht noch einmal generiert wird. Zu diesem Zweck ist es erforderlich, dass der präsentierte Ansatz weiterentwickelt wird, sodass gegebenes Domänenwissen vom System erlernt werden kann, indem es gespeichert und für nachfolgende Analysen wiederverwendet wird. Hierzu können automatisierte Techniken aus dem Bereich des *Maschinellen Lernens* eingesetzt werden [Alp16]. Diese wären auch für große Datenanalysen im Kontext von gewachsenen Technologiearchitekturen verwendbar [ABC+16]. Hierdurch kann der manuelle Aufwand unseres Ansatzes weiter reduziert werden, indem die wiederholte Eingabe von gleichem Domänenwissen verhindert oder zumindest verringert wird.

Einbindung weiterer Layer der Unternehmensarchitektur: Während sich der entwickelte Gesamtansatz auf die Analyse von Technologiearchitekturen fokussiert, bleiben andere Layer einer betrachteten Unternehmensarchitektur unberücksichtigt. In Gesprächen mit Experten hat sich aber gezeigt, dass die Einbindung von Elementen weiterer Architekturlayer förderlich für die Entscheidungsfindung zur Variabilitätsreduzierung in Technologiearchitekturen sein kann. So ist es denkbar, dass Elemente wie *Geschäftsobjekte*, *Geschäftsprozesse* und *Geschäftsstandorte* einer *Geschäftsarchitektur* die businessorientierte Analyse zur Ableitung von Restrukturierungsempfehlungen unterstützen können. Zudem wäre es auch vorstellbar, dass Elemente wie *Applikationen*, *Datenobjekte* und *Schnittstellen* einer *Applikationsarchitektur* noch präzisere Ergebnisse in der technologieorientierten Analyse zur Bestimmung der technischen Machbarkeit ermöglichen können. Hierzu müssten die entwickelten Modelle und Techniken unseres Verfahrens erweitert werden, um auch solche Daten verarbeiten zu können. Insgesamt würde diese Erweiterung des bestehenden Ansatzes dazu führen, dass noch mehr technologiebezogene Informationen automatisiert verarbeitet werden können. Dies ermöglicht noch präzisere Ergebnisse, wodurch Domänenexperten mit Hilfe unseres Ansatz in der Entscheidungsfindung noch besser unterstützen werden können.

Literatur

[ABC+16] Abadi, M. ; Barham, P. ; Chen, J. ; Chen, Z. ; Davis, A. ; Dean, J.: TensorFlow: A system for large-scale machine learning. In: *Proc. des Symposium on Operating Systems Design and Implementation (OSDI)*, USENIX, 2016, S. 265–283 201

[ABCB15] Antunes, G. ; Barateiro, J. ; Caetano, A. ; Borbinha, J.: Analysis of Federated Enterprise Architecture Models. In: *Proc. der European Conference on Information Systems (ECIS)*, AIS, 2015 4, 67

[ABM+14a] Antunes, G. ; Bakhshandeh, M. ; Mayer, R. ; Borbinha, J. ; Caetano, A.: Ontology-Based Enterprise Architecture Model Analysis. In: *Proc. des Symposium On Applied Computing (SAC)*, ACM, 2014, S. 1420–1422 67

[ABM+14b] Antunes, G. ; Bakhshandeh, M. ; Mayer, R. ; Borbinha, J. ; Caetano, A.: Using Ontologies for Enterprise Architecture Integration and Analysis. In: *Complex Systems Informatics and Modeling Quarterly* 1 (2014), Nr. 1, S. 1–23 67

[ABR09] Akella, J. ; Buckow, H. ; Rey, S.: *IT Architecture: Cutting Costs and Complexity*. `https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/it-architecture-cutting-costs-and-complexity`. Version: 2009. – Eingesehen am 28.08.2018 113

[ACB+13] Antunes, G. ; Caetano, A. ; Bakhshandeh, M. ; Mayer, R. ; Borbinha, J.: Using Ontologies to Integrate Multiple Enterprise Architecture Domains. In: *Proc. des Workshop on Business and IT Alignment (BITA)*, Springer, 2013, S. 61–72 67

[ACM+11] Acher, M. ; Cleve, P. A.and C. A.and Collet ; Merle, P. ; Duchien, L. ; Lahire, P.: Reverse Engineering Architectural Feature Models. In: *Proc. der European Conference of Software Architecture (ECSA)*, Springer, 2011 (Lecture Notes in Computer Science), S. 220–235 3, 70

[AG14] Albayrak, C. A. ; Gadatsch, A.: Komplexitätsmanagement als Instrument des IT-Controllings. In: *Controlling* 26 (2014), Nr. 12, S. 680–685 113

[Alp16] Alpaydin, E.: *Machine Learning: The New AI*. Cambridge : MIT Press, 2016 201

[ASML12] Ahlemann, F. ; Stettiner, E. ; Messerschmidt, M. ; Legner, C.: *Strategic Enterprise Architecture Management – Challenges, Best Practices, and Future Developments*. Berlin, Heidelberg : Springer, 2012 1, 2, 22, 24

[BAHA16] Beese, J. ; Aier, S. ; Haki, M. K. ; Aleatrati Khosroshahi, P.: Drivers and Effects of Information Systems Architecture Complexity: A Mixed-Methods Study. In: *Proc. der European Conference on Information Systems (ECIS)*, AIS, 2016 2

[Bat05] Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, Springer, 2005, S. 7–20 16

[BCS10] Babar, M. A. ; Chen, L. ; Shull, F.: Managing Variability in Software Product Lines. In: *IEEE Software* 27 (2010), Nr. 3, S. 89–94 11, 13, 68

[Bee14] Beetz, K. R.: *Wirkung von IT-Governance auf IT-Komplexität in Unternehmen, Dissertation*. Wiesbaden : Springer Gabler, 2014 65

[BELM08] Buckl, S. ; Ernst, A. M. ; Lankes, J. ; Matthes, F.: Enterprise Architecture Management Pattern Catalog – Version 1.0 / TU München. 2008. – Forschungsbericht 111

[Ben14] Bente, S.: Kollaborative Enterprise-Architektur — Managementwerkzeug für komplexe IT-Systeme. In: Schoeneberg, K.-P. (Hrsg.): *Komplexitätsmanagement in Unternehmen*. Wiesbaden : Springer, 2014, S. 187–223 1

[Ber06] Berkhin, P.: A Survey of Clustering Data Mining Techniques. In: Kogan, J. (Hrsg.) ; Nicholas, C. (Hrsg.) ; Teboulle, M. (Hrsg.): *Grouping Multidimensional Data*. Berlin, Heidelberg : Springer, 2006, S. 25–71 44, 200

[BHS07] Buschmann, F. ; Henney, K. ; Schmidt, D. C.: *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*. Chichester : Wiley, 2007 157

[BJS13] Buschle, M. ; Johnson, P. ; Shahzad, K.: The Enterprise Architecture Analysis Tool – Support for the Predictive, Probabilistic Architecture Modeling Framework. In: *Proc. der Americas Conference on Information Systems (AMCIS)*, AIS, 2013, S. 3350–3364 4, 67, 114

[BK11] Beetz, K. R. ; Kolbe, L. M.: Towards Managing IT Complexity: An IT Governance Framework to Measure Business-IT Responsibility Sharing and Structural IT Organization. In: *Proc. der Americas Conference on Information Systems (AMCIS)*, AIS, 2011, S. 832–843 65

[BKAV17] Beese, J. ; Khosroshahi, P. A. ; Aier, S. ; Volkert, S.: Komplexität von IT-Landschaften. In: *Wirtschaftsinformatik & Management* 9 (2017), Nr. 2, S. 40–46 1, 112

[BLNS12] Binz, T. ; Leymann, F. ; Nowak, A. ; Schumm, D.: Improving the Manageability of Enterprise Topologies Through Segmentation, Graph Transformation, and Analysis Strategies. In: *Proc. der Intl. Enterprise Distributed Object Computing Conference (EDOC)*, IEEE, 2012, S. 61–70 4, 97, 114

[BLS11] Biedermann, K. ; Lüttich, M. ; Scherdin, A.: Wertorientierte IT-Transformation durch Bebauungsplanung. In: Lang, M. (Hrsg.) ; Amberg, M. (Hrsg.): *Erfolgsfaktor IT-Management: So steigern Sie den Wertbeitrag Ihrer IT*. Düsseldorf : Symposion-Publ., 2011, S. 367–404 4, 111

[BMS09] Buckl, S. ; Matthes, F. ; Schweda, C. M.: A Viable System Perspective on Enterprise Architecture Management. In: *Proc. der Intl. Conference on Systems, Man and Cybernetics (SMC)*, IEEE, 2009, S. 1483–1488 66

[Bos00] Bosch, J.: *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. New York : ACM Press/Addison-Wesley, 2000 11

[BPB15] Bakhshandeh, M. ; Pesquita, C. ; Borbinha, J.: Ontology Matching Techniques for Enterprise Architecture Models. In: *Proc. des Intl. Workshop on Ontology Matching (OM)*, CEUR-WS, 2015, S. 236–237 66

[BPB16] Bakhshandeh, M. ; Pesquita, C. ; Borbinha, J.: An Ontological Matching Approach for Enterprise Architecture Model Analysis. In: *Proc. der Intl. Conference on Business Information Systems (BIS)*, Springer, 2016, S. 315–326 66

[Bro87] Brooks, F.: No Silver Bullet – Essence and Accidents of Software Engineering. In: *Computer* 20 (1987), Nr. 4, S. 10–19 15

[BSB+17] Bhat, M. ; Shumaiev, K. ; Biesdorf, A. ; Hohenstein, U. ; Hassel, M. ; Matthes, F.: An Ontology-Based Approach for Software Architecture Recommendations. In: *Proc. der Americas Conference on Information Systems (AMCIS)*, AIS, 2017, S. 1–10 4, 114

[BUF+10] Buschle, M. ; Ullberg, J. ; Franke, U. ; Lagerström, R. ; Sommestad, T.: A Tool for Enterprise Architecture Analysis Using the PRM Formalism. In: *Proc. der Intl. Conference on on Advanced Information Systems Engineering (CAiSE)*, Springer, 2010, S. 108–121 152

[Bus14] Buschle, M.: *Tool Support for Enterprise Architecture Analysis, Dissertation.* Stockholm : KTH, Royal Institute of Technology, 2014 4, 114

[BZ18] Busch, N. R. ; Zelewski, A. ; Kosiuczenko, P. (Hrsg.) ; Madeyski, L. (Hrsg.): *Enterprise Architecture Modifiability Analysis.* 2018 5, 152

[CA05] Czarnecki, K. ; Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: *Proc. der Intl. Conference on Generative Programming and Component Engineering (GPSE)*, Springer, 2005, S. 422–437 18

[CE00] Czarnecki, K. ; Eisenecker, U. W.: *Generative Programming: Methods, Tools, and Applications.* Boston : Addison Wesley Longman, 2000 12, 15, 16

[CHS10] Clarke, D. ; Helvensteijn, M. ; Schaefer, I.: Abstract Delta Modeling. In: *Proc. der Intl. Conference on Generative Programming and Component Engineering (GPSE)*, ACM, 2010, S. 13–22 19

[CN07] Clements, Paul C. ; Northrop, Linda M.: *Software Product Lines: Practices and Patterns.* Boston : Addison Wesley Longman, 2007 11, 12

[CV09] Corneliussen, L. ; Voelter, M.: Kurzer Rede, langer Sinn – Domänenspezifische Sprachen als Mittel zur Abstraktion in der Software-Entwicklung. In: *dotnetpro* 97 (2009), Nr. 5 29, 30

[CvL10] Chuang, C.-H. ; van Loggerenberg, J. ; Lotriet, H.: Towards Improving Enterprise Architecture Decision-Making Through Service-Dominant Logic. In: *Proc. der Pacific Asia Conference on Information Systems (PACIS)*, AIS, 2010, S. 1263–1273 5, 154

[Cza10] Czarnecki, K.: Variability Modeling: State of the Art and Future Directions. In: *Proc. des Intl. Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, ICB-Research Report No. 37, Universität Duisburg Essen, 2010, S. 11 15

[Day95] Dayal, U.: Ten Years of Activity in Active Database Systems: What Have We Accomplished? In: *Proc. des Intl. Workshop on Active and Real-Time Database Systems (ARTDB)*, Springer, 1995, S. 3–22 76

[Der09] Dern, G.: *Management von IT-Architekturen.* Wiesbaden : Vieweg+Teubner, 2009 1, 22, 23

[Der11] Dern, G.: *Integrationsmanagement in der Unternehmens-IT: Systemtheoretisch fundierte Empfehlungen zur Gestaltung von IT-Landschaft und IT-Organisation.* Wiesbaden : Vieweg+Teubner, 2011 65

[DJ09] Dern, G. ; Jung, R.: IT-Architektur-Governance auf Basis von Kennzahlen zur Komplexitätsmessung. In: *Controlling* 21 (2009), Nr. 12, S. 669–672 65

[DKK+12] Durdik, Z. ; Klatt, B. ; Koziolek, H. ; Krogmann, K. ; Stammel, J. ; Weiss, R.: Sustainability Guidelines for Long-Living Software Systems. In: *Proc. der Intl. Conference on Software Maintenance (ICSM)*, IEEE, 2012, S. 517–526 1

[DS15] Damyanov, I. ; Sukalinska, M.: Domain Specific Languages in Practice. In: *International Journal of Computer Applications* 115 (2015), Nr. 2, S. 42–45 29

[DUA14] Dittes, S. ; Urbach, N. ; Ahlemann, F.: IT-Standardisierung – vom Lippenbekenntnis zu nachhaltigem Nutzen. In: *Wirtschaftsinformatik & Management* 6 (2014), Nr. 4, S. 30–39 111

[Dur08] Durst, M.: *Wertorientiertes Management von IT-Architekturen, Dissertation.* Wiesbaden : Deutscher Universitätsverlag, 2008 vii, 1, 4, 25, 111

[EFJ+09] Ekstedt, M. ; Franke, U. ; Johnson, P. ; Lagerström, R. ; Sommestad, T. ; Ullberg, J. ; Buschle, M.: A Tool for Enterprise Architecture Analysis of Maintainability. In: *Proc. der European Conference on Software Maintenance and Reengineering (CSMR)*, IEEE, 2009, S. 327–328 152

[EHH+08] Engels, G. ; Hess, A. ; Humm, B. ; Juwig, O. ; Lohmann, M. ; Richter, J. P. ; Voss, M. ; Willkomm, J.: *Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten.* Heidelberg : dpunkt.verlag, 2008 22

[Ell94] Ellram, L.: A Taxonomy of Total Cost of Ownership Models. In: *Journal of Business Logistics* 15 (1994), Nr. 1, S. 171–191 76

[EN16] Eskandari, M. ; Nabiollahi, A.: A Method for Prioritizing Qualitative Scenarios in Evaluating Enterprise Architecture Using Non-Dominated Sorting Genetic Algorithm II. In: *Int. Journal of Computer Science and Information Technology* 8 (2016), Nr. 6, S. 29–38 5, 151, 152

[Eve11] Everitt, B.: *Cluster Analysis*. Chichester : Wiley, 2011 (Wiley series in probability and statistics) 44

[FAHC15] Font, J. ; Arcega, L. ; Haugen, Ø. ; Cetina, C.: Building Software Product Lines from Conceptualized Model Patterns. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, ACM, 2015, S. 46–55 68, 70

[FBHC15] Font, J. ; Ballarín, M. ; Haugen, Ø. ; Cetina, C.: Automating the Variability Formalization of a Model Family by Means of Common Variability Language. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, ACM, 2015, S. 411–418 68, 70

[FP10] Fowler, M. ; Parsons, R.: *Domain-Specific Languages*. Boston : Addison-Wesley, 2010 29

[FP14] Fauscette, M. ; Perry, R.: Simplifying IT to Drive Better Business Outcomes and Improved ROI: Introducing the IT Complexity Index / International Data Corporation. 2014. – White Paper 4, 113

[GAE+16] Govin, B. ; Anquetil, N. ; Etien, A. ; Ducasse, S. ; Monegier, A.: How Can We Help Software Rearchitecting Efforts? Study of an Industrial Case. In: *Proc. der Intl. Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2016, S. 509–518 3, 117

[GAWM11] Galster, M. ; Avgeriou, P. ; Weyns, D. ; Männistö, T.: Variability in Software Architecture: Current Practice and Challenges. In: *ACM SIGSOFT Software Engineering Notes* 36 (2011), Nr. 5, S. 30–32 2, 13

[GD13] Grebe, M. ; Danke, E.: Simplify IT – Six Ways to Reduce Complexity / The Boston Consulting Group. 2013. – White Paper 113

[GHK17] Gruber, M. ; Hoffmann, J. ; Karla, J.: Methoden zum Management der IT-Komplexität. In: *INFORMATIK – Beiträge der Jahrestagung der Gesellschaft für Informatik*. Bonn : GI, 2017, S. 1601–1615 110, 111, 114

[GKR+08] Grönniger, H. ; Krahn, H. ; Rumpe, B. ; Schindler, M. ; Völkel, S.: MontiCore: A Framework for the Development of Textual Domain Specific Languages. In: *Proc. der Intl. Conference on Software Engineering (ICSE)*, ACM, 2008, S. 925–926 29

[Gom05] Gomaa, H.: Designing Software Product Lines with UML. In: *Proc. des Software Engineering Workshop (SEW)*, IEEE, 2005, S. 160–216 18

[GWT+14] Galster, M. ; Weyns, D. ; Tofan, D. ; Michalik, B. ; Avgeriou, P.: Variability in Software Systems – A Systematic Literature Review.

In: *IEEE Transactions on Software Engineering* 40 (2014), Nr. 3, S. 282–306 13, 68

[GZW+17] Galster, M. ; Zdun, U. ; Weyns, D. ; Rabiser, R. ; Zhang, B. ; Goedicke, M. ; Perrouin, G.: Variability and Complexity in Software Design – Towards a Research Agenda. In: *ACM SIGSOFT Software Engineering Notes* 41 (2017), Nr. 6, S. 27–30 4, 14, 15

[Han10] Hanschke, I.: *Strategic IT Management – A Toolkit for Enterprise Architecture Management*. Berlin, Heidelberg : Springer, 2010 1, 111

[Han11] Hanschke, I.: Beherrschen der IT-Komplexität mithilfe von EAM. In: *Wirtschaftsinformatik & Management* 3 (2011), Nr. 4, S. 66–71 2, 4, 111, 113

[HHH09] Holtschke, B. ; Heier, H. ; Hummel, T.: *Quo vadis CIO?* Berlin, Heidelberg : Springer, 2009 1, 2

[HMPO+08] Haugen, Ø. ; Møller-Pedersen, B. ; Oldevik, J. ; Olsen, G. ; Svendsen, A.: Adding Standardized Variability to Domain Specific Languages. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, IEEE, 2008, S. 139–148 15

[HR04] Harel, D. ; Rumpe, B.: Meaningful Modeling: What's the Semantics of Semantics? In: *IEEE Computer* 37 (2004), Nr. 10, S. 64–72 29

[HRPM13] Hauder, M. ; Roth, S. ; Pigat, S. ; Matthes, F.: A Configurator for Visual Analysis of Enterprise Architectures. In: *Proc. der Intl. Conference on Model Driven Engineering Languages and Systems (MODELS)*, Springer, 2013 (Lecture Notes in Computer Science) 67

[ISO96] International Organization for Standardization: Information Technology – Syntactic Metalanguage – Extended BNF. Geneva, 1996. – Standard 30

[ISO11] ISO/IEC/IEEE: Systems and Software Engineering - Architecture Description. 42010-2011. Geneva and New York : ISO, IEC, IEEE, 2011. – Standard Norm 22, 24

[JESP04] Johnson, P. ; Ekstedt, M. ; Silva, E. ; Plazaola, L.: Using Enterprise Architecture for CIO Decision-Making: On the Importance of Theory. In: *Proc. der Conference on Systems Engineering Research (CSER)*, 2004 154

[JJSU07] Johnson, P. ; Johansson, E. ; Sommestad, T. ; Ullberg, J.: A Tool for Enterprise Architecture Analysis. In: *Proc. der Intl. Enterprise Distributed Object Computing Conference (EDOC)*, IEEE, 2007, S. 142–153 152

[JKSZ15a] Jugel, D. ; Kehrer, S. ; Schweda, C. M. ; Zimmermann, A.: A Decision-Making Case for Collaborative Enterprise Architecture Engineering. In: *INFORMATIK – Beiträge der Jahrestagung der Gesellschaft für Informatik*, GI, 2015, S. 865–379 153

[JKSZ15b] Jugel, D. ; Kehrer, S. ; Schweda, C. M. ; Zimmermann, A.: Providing EA Decision Support for Stakeholders by Automated Analyses. In: *Digital Enterprise Computing*, GI, 2015, S. 151–162 153

[JPD09] Javanbakht, M. ; Pourkamali, M. ; Derakhshi, M. R.: A New Method for Enterprise Architecture Assessment and Decision-Making about Improvement or Redesign. In: *Proc. der Intl. Conference on Computing in the Global Information Technology (ICCGI)*, IEEE, 2009, S. 69–76 5, 153

[JRSS08] Javanbakht, M. ; Rezaie, R. ; Shams, F. ; Seyyedi, M.: A New Method for Decision Making and Planning in Enterprises. In: *Proc. der Intl. Conference on Information and Communication Technologies, from Theory to Applications (ICTTA)*, IEEE, 2008, S. 1–5 153, 154

[JSZ15] Jugel, D. ; Schweda, C. M. ; Zimmermann, A.: Modeling Decisions for Collaborative Enterprise Architecture Engineering. In: *Proc. der Intl. Conference on on Advanced Information Systems Engineering (CAiSE)*, Springer, 2015, S. 351–362 5, 153, 154

[JUB+13] Johnson, P. ; Ullberg, J. ; Buschle, M. ; Franke, U. ; Khurram, S.: P2 AMF: Predictive, Probabilistic Architecture Modeling Framework. In: *Proc. der Intl. IFIP Working Conference on Enterprise Interoperability (IWEI)*, Springer, 2013, S. 104–117 114

[KCH+90] Kang, K. C. ; Cohen, S. G. ; Hess, J. A. ; Novak, W. E. ; Peterson, A. S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study / Carnegie-Mellon University Software Engineering Institute. 1990 (CMU/SEI-90-TR-021). – Forschungsbericht 11, 15, 16

[KDO14] Kaestner, C. ; Dreiling, A. ; Ostermann, K.: Variability Mining: Consistent Semi-automatic Detection of Product-Line Features. In: *IEEE Transactions on Software Engineering* 40 (2014), Nr. 1, S. 67–82 20

[Kel12] Keller, W.: *IT-Unternehmensarchitektur – Von der Geschäftsstrategie zur optimalen IT-Unterstützung*. Heidelberg : dpunkt.verlag, 2012 22, 111, 139

[Kha11] Khayami, R.: Qualitative Characteristics of Enterprise Architecture. In: *Procedia Computer Science* 3 (2011), Nr. 1, S. 1277–1282 151

[KHSM15] Khosroshahi, P. A. ; Hauder, M. ; Schneider, A. W. ; Matthes, F.: Enterprise Architecture Management Pattern Catalog – Version 2.0 / TU München. 2015. – Forschungsbericht 111

[KKC00] Kazman, R. ; Klein, M. ; Clements, P.: ATAM: Method for Architecture Evaluation / Software Engineering Institute, Pittsburgh, USA. 2000 (CMU/SEI-2000-TR-004). – Forschungsbericht 151

[Kra10] Krahn, H.: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering, Dissertation. In: Rumpe, B. (Hrsg.): *Aachener Informatik Berichte Software Engineering*. Aachen : RWTH Aachen, 2010 30

[Kru01] Krueger, C. W.: Easing the Transition to Software Mass Customization. In: *Proc. des Intl. Workshop on Software Product-Family Engineering (PFE)*, Springer, 2001, S. 282–293 14, 20

[KRV08a] Krahn, H. ; Rumpe, B. ; Völkel, S.: Mit Sprachbaukästen zur schnelleren Softwareentwicklung: Domänenspezifische Sprachen modular entwickeln. In: *OBJEKTspektrum* (2008), Nr. 4, S. 42–46 30

[KRV08b] Krahn, H. ; Rumpe, B. ; Völkel, S.: MontiCore: Modular Development of Textual Domain Specific Languages. In: *Proc. der Intl. Conference on Objects, Components, Models and Patterns (TOOLS-Europe)*, Springer, 2008 (LNBIP), S. 297–315 29

[KSD14] Karimi, M. ; Sharafi, S. M. ; Dehkordi, M. N.: A New Approach Based on Genetic Algorithm for Prioritizing Quality Scenarios in Enterprise Architecture Evaluation. In: *Int. Journal of Computer Science Engineering* 3 (2014), Nr. 1, S. 21–31 5, 151, 152

[KWAB94] Kazman, R. ; Webb, M. ; Abow, G. ; Bass, L.: SAAM: A Method for Analyzing the Properties of Software Architectures. In: *Proc. der Intl. Conference on Software Engineering (ICSE)*, ACM, 1994, S. 81–90 152

[Lag07] Lagerström, R.: Analyzing System Maintainability using Enterprise Architecture Models. In: *Journal of Enterprise Architecture* 3 (2007), Nr. 4, S. 33–41 152

[LB15] Lakhrouit, J. ; Baina, K.: Evaluating Complexity of Enterprise Architecture Components Landscapes. In: *Proc. der Intl. Conference on Intelligent Systems: Theories and Applications (SITA)*, IEEE, 2015, S. 415–419 64

[LB16] Lakhrouit, J. ; Baina, K.: Enterprise Architecture Complexity Component Based on Archimate Language. In: *Proc. des Intl. Symposium on Ubiquitous Networking (UNet)*, Springer, 2016, S. 535–546 65

[LBMA14] Lagerström, R. ; Baldwin, C. ; MacCormack, A. ; Aier, S.: Visualizing and Measuring Enterprise Application Architecture: An Exploratory Telecom Case. In: *Proc. der Hawaii Intl. Conference on System Sciences (HICSS)*, IEEE, 2014, S. 3847–3856 4, 64

[Leh80] Lehman, M. M.: Programs, Life Cycles, and Laws of Software Evolution. In: *Proceedings of the IEEE* 68 (1980), Nr. 9, S. 1060–1076 14

[LFJU09] Lagerström, R. ; Franke, U. ; Johnson, P. ; Ullberg, J.: A Method for Creating Enterprise Architecture Metamodels – Applied to Systems Modifiability Analysis. In: *Int. Journal of Computer Science and Applications* 6 (2009), Nr. 5, S. 89–120 152

[LGv+16] Lapalme, J. ; Gerber, A. ; van der Merwe, A. ; Zachman, J. ; De Vries, M. ; Hinkelmann, K.: Exploring the Future of Enterprise Architecture: A Zachman Perspective. In: *Computers in Industry* 79 (2016), Nr. 5, S. 103–113 1

[Liu02] Liu, S.: A Practical Framework for Discussing IT Infrastructure. In: *IEEE IT Professional* 4 (2002), Nr. 4, S. 14–21 vii, 25, 26

[LJ08] Lagerström, R. ; Johnson, P.: Using Architectural Models to Predict the Maintainability of Enterprise Systems. In: *Proc. der European Conference on Software Maintenance and Reengineering (CSMR)*, IEEE, 2008, S. 248–252 152

[LJH10] Lagerström, R. ; Johnson, P. ; Höök: Architecture Analysis of Enterprise Systems Modifiability – Models, Analysis, and Validation. In: *Journal of Systems and Software* 83 (2010), Nr. 8, S. 1387–1403 5, 152

[LLHE17] Linsbauer, L. ; Lopez-Herrejon, R. E. ; Egyed, A.: Variability Extraction and Modeling for Product Variants. In: *Software and Systems Modeling* 16 (2017), Nr. 4, S. 1179–1199 3, 14, 20, 68

[LRR03] Lichtenegger, R. ; Rohloff, M. ; Rosauer, B.: Beschreibung von Unternehmensarchitekturen: Sichten und Abhängigkeiten am Beispiel der IT-Infrastrukturarchitektur. In: *INFORMATIK – Beiträge der Jahrestagung der Gesellschaft für Informatik*, GI, 2003, S. 426–434 24

[LZ13] Lytra, I. ; Zdun, U.: Supporting Architectural Decision Making for Systems-of-Systems Design Under Uncertainty. In: *Proc. des Intl. Workshop on Software Engineering for Systems-of-Systems (SESoS)*, ACM, 2013, S. 43–46 5, 154, 155

[MA02] Muthig, D. ; Atkinson, C.: Model-Driven Product Line Architectures. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, ACM, 2002, S. 110–129 15

[Mat08] Matthes, F.: Softwarekartographie – Anwendungslandschaften verstehen und gestalten. In: *Informatik Spektrum* 31 (2008), Nr. 6, S. 527–536 27

[Mat15] Maturity GmbH: *Studie: IT-Komplexität 2015.* `https://www.maturity.com/de/pressemitteilung/studie-it-komplexitaet-2015.html`. Version: 2015. – Eingesehen am 13.11.2018 1

[May07] Mayring, P.: *Qualitative Inhaltsanalyse: Grundlagen und Techniken.* Weinheim, Basel : Belitz, 2007 185

[Moc09] Mocker, M.: What Is Complex About 273 Applications? Untangling Application Architecture Complexity in a Case of European Investment Banking. In: *Proc. der Hawaii Intl. Conference on System Sciences (HICSS)*, IEEE, 2009, S. 3615–3628 65

[MZB+15] Martinez, J. ; Ziadi, T. ; Bissyandé, T. F. ; Klein, J. ; Traon, Y. l.: Automating the Extraction of Model-Based Software Product Lines from Model Variants. In: *Proc. der Intl. Conference on Automated Software Engineering (ASE)*, IEEE, 2015, S. 396–406 3, 4, 14, 20, 69, 70

[MZKT14] Martinez, J. ; Ziadi, T. ; Klein, J. ; Traon, Y. l.: Identifying and Visualising Commonality and Variability in Model Variants. In: *Proc. der European Conference on Modelling Foundations and Applications (ECMFA)*, Springer, 2014 (Lecture Notes in Computer Science), S. 117–131 69

[NARS16] Nikpay, F. ; Ahmad, R. ; Rouhani, B. D. ; Shamshirband, S.: A Systematic Review on Post-Implementation Evaluation Models of Enterprise Architecture Artefacts. In: *Information Systems Frontiers* 18 (2016), Nr. 6, S. 1–20 151

[NBE14] Närman, P. ; Buschle, M. ; Ekstedt, M.: An Enterprise Architecture Framework for Multi-Attribute Information Systems Analysis. In: *Software and Systems Modeling* 13 (2014), Nr. 3, S. 1085–1116 67

[NSC+07] Nejati, S. ; Sabetzadeh, M. ; Chechik, M. ; Easterbrook, S. ; Zave, P.: Matching and Merging of Statecharts Specifications. In: *Proc. der Intl. Conference on Software Engineering (ICSE)*, IEEE, 2007, S. 54–64 3, 68, 70

[NSC+12] Nejati, S. ; Sabetzadeh, M. ; Chechik, M. ; Easterbrook, S. ; Zave, P.: Matching and Merging of Variant Feature Specifications. In: *IEEE Transactions on Software Engineering* 38 (2012), Nr. 6, S. 1355–1375 68

[NvP11] Nakakawa, A. ; van Bommel, P. ; Proper, H. A.: Definition and Validation of Requirements for Collaborative Decision-Making in Enterprise Architecture Creation. In: *Int. Journal of Cooperative Information Systems* 20 (2011), Nr. 1, S. 83–136 5, 153, 154

[OJK+13] Österlind, M. ; Johnson, P. ; Karnati, K. ; Lagerström, R. ; Välja, M.: Enterprise Architecture Evaluation Using Utility Theory. In: *Proc. der Intl. Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, IEEE, 2013, S. 347–351 5, 151, 152

[PBv05] Pohl, K. ; Böckle, G. ; van der Linden, F. J.: *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Heidelberg : Springer, 2005 12, 13, 14, 15

[PdL+13] Plataniotis, G. ; de Kinderen, S. ; Linden, D. ; Greefhorst, D. ; Proper, H. A.: An Empirical Evaluation of Design Decision Concepts in Enterprise Architecture. In: *Proc. der Conference on The Practice of Entrprise Modeling (PoEM)*, Springer, 2013, S. 24–38 153

[PdP13] Plataniotis, G. ; de Kinderen, S. ; Proper, H. A.: Relating Decisions in Enterprise Architecture Using Decision Design Graphs. In: *Proc. der Intl. Enterprise Distributed Object Computing Conference (EDOC)*, IEEE, 2013, S. 139–146 153

[PdP14] Plataniotis, G. ; de Kinderen, S. ; Proper, H. A.: EA Anamnesis: An Approach for Decision Making Analysis in Enterprise Architecture. In: *Int. Journal of Information System Modeling and Design* 5 (2014), Nr. 3, S. 75–95 5, 153, 154, 155

[PST12] Pande, S. R. ; Sambare, S. S. ; Thakre, V. M.: Data Clustering Using Data Mining Techniques. In: *Int. Journal of Advanced Research in Computer and Communication Engineering* 1 (2012), Nr. 8, S. 494–499 200

[RC12] Rubin, J. ; Chechik, M.: Combining Related Products into Product Lines. In: *Proc. der Intl. Conference on Fundamental Approaches to Software Engineering (FASE)* Bd. 7212, Springer, 2012 (Lecture Notes in Computer Science), S. 285–300 68, 69, 70

[RC13a] Rubin, J. ; Chechik, M.: N-Way Model Merging. In: *Proc. des Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESECFSE)*, ACM, 2013, S. 301–311 69, 70

[RC13b] Rubin, J. ; Chechik, M.: Quality of Merge-Refactorings for Product Lines. In: *Proc. der Intl. Conference on Fundamental Approaches to*

Software Engineering (FASE) Bd. 7793, Springer, 2013 (Lecture Notes in Computer Science), S. 83–98 68

[RDGB08] Rabiser, R. ; Dhungana, D. ; Grünbacher, P. ; Burgstaller, B.: Value-Based Elicitation of Product Line Variability: An Experience Report. In: *Proc. des Intl. Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, ICB Research Report, 2008, S. 73–79 20

[RHZ+13] Roth, S. ; Hauder, M. ; Zec, M. ; Utz, A. ; Matthes, F.: Empowering Business Users to Analyze Enterprise Architectures: Structural Model Matching to Configure Visualizations. In: *Proc. der Intl. Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, IEEE, 2013, S. 352–360 67

[Ros03a] Ross, J. W.: Creating a Strategic IT Architecture Competency: Learning in Stages. In: *MIS Quarterly Executive* 2 (2003), Nr. 1, S. 31–43 23

[Ros03b] Ross, R. G.: *Principles of the Business Rule Approach.* Addison-Wesley, 2003 75

[RS09] Razavi Davoudi, M. ; Shams Aliee, F.: A new AHP-based Approach Towards Enterprise Architecture Quality Attribute Analysis. In: *Proc. der Intl. Conference on Research Challenges in Information Science (RCIS)*, IEEE, 2009, S. 333–342 151

[RS12] Razavi Davoudi, M. ; Sheikhvand, K.: An Approach towards Enterprise Architecture Analysis using AHP and Fuzzy AHP. In: *Int. Journal of Machine Learning and Computing* 2 (2012), Nr. 1, S. 46–51 151

[RV96] Real, R. ; Vargas, J. M.: The Probabilistic Basis of Jaccard's Index of Similarity. In: *Systematic Biology* 45 (1996), Nr. 3, S. 380–385 101

[SBB+10] Schaefer, I. ; Bettini, L. ; Bono, V. ; Damiani, F. ; Tanzarella, N.: Delta-Oriented Programming of Software Product Lines. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, Springer, 2010 (Lecture Notes in Computer Science), S. 77–91 19

[Sch02] Scheer, A.-W.: *ARIS — Vom Geschäftsprozess zum Anwendungssystem.* Berlin, Heidelberg : Springer, 2002 76

[Sch09] Schmidt, C.: *Management komplexer IT-Architekturen, Dissertation.* Wiesbaden : Gabler, 2009 1, 24

[Sch10] Schaefer, I.: Variability Modelling for Model-Driven Development of Software Product Lines. In: *Proc. des Intl. Workshop on Variability*

Modelling of Software-Intensive Systems (VaMoS), ACM, 2010, S. 85–92 19

[Sch16] Schneider, A. W.: *Decision Support for Application Landscape Diversity Management, Dissertation.* München : TU München, 2016 1, 22, 66

[SD10] Schaefer, I. ; Damiani, F.: Pure Delta-Oriented Programming. In: *Proc. des Intl. Workshop on Feature Oriented Software Development (FOSD)*, ACM, 2010, S. 49–56 19

[Sei17] Seidl, C.: *Integrated Management of Variability in Space and Time in Software Families, Dissertation.* Dresden : TU Dresden, 2017 11, 13, 18, 45

[SGM15] Schneider, A. W. ; Gschwendter, A. ; Matthes, F.: IT Architecture Standardization Survey / Technische Universität München. 2015. – Forschungsbericht 22

[Sha48] Shannon, C. E.: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), Nr. 3, S. 379–423 64

[SHGZ17] Schuh, G. ; Hoffmann, J. ; Gruber, M. ; Zeller, V.: Managing IT Complexity in the Manufacturing Industry – An Agenda for Action. In: *Journal of Systemics, Cybernetics and Informatics* 15 (2017), Nr. 2, S. 61–65 110, 111

[SKR13] Sunkle, S. ; Kulkarni, V. ; Roychoudhury, S.: Analyzing Enterprise Models Using Enterprise Architecture-Based Ontology. In: *Proc. der Intl. Conference on Model Driven Engineering Languages and Systems (MODELS)*, Springer, 2013 (Lecture Notes in Computer Science), S. 622–638 4, 67

[SLJ+05] Simonsson, M. ; Lindstrom, S. ; Johnson, P. ; Nordstrom, L. ; Grundbäck, J. ; Wijnbladh, O.: Scenario-Based Evaluation of Enterprise Architecture – A Top-Down Approach for Chief Information Officer Decision Making. In: *Proc. der Intl. Conference on Enterprise Information Systems (ICEIS)*, SCITEPRESS, 2005, S. 130–137 154

[SM03] Schneberger, S. L. ; McLean, E. R.: The Complexity Cross: Implications for Practice. In: *Communications of the ACM* 46 (2003), Nr. 9, S. 216–225 4, 63

[SM14] Schneider, A. W. ; Matthes, F.: Unternehmensarchitekturgestütztes Controlling zur Beherrschung der IT-Komplexität. In: *Controlling* 26 (2014), Nr. 12, S. 694–699 65

[SM15] Schneider, A. W. ; Matthes, F.: Evolving the EAM Pattern Language. In: *Proc. der European Conference on Pattern Languages of Programs (EuroPLoP)*, ACM, 2015, S. 45:1–45:11 4, 111

[SRC+12] Schaefer, I. ; Rabiser, R. ; Clarke, D. ; Bettini, L. ; Benavides, D. ; Botterweck, G. ; Pathak, A. ; Trujillo, S. ; Villela, K.: Software Diversity: State of the Art and Perspectives. In: *Int. Journal on Software Tools for Technology Transfer* 14 (2012), Nr. 5, S. 477–495 11, 13, 15, 16, 17, 18, 19

[SRSM15] Schneider, A. W. ; Reschenhofer, T. ; Schütz, A. ; Matthes, F.: Empirical Results for Application Landscape Complexity. In: *Proc. der Hawaii Intl. Conference on System Sciences (HICSS)*, IEEE, 2015, S. 4079–4088 65

[SSA14] Seidl, C. ; Schaefer, I. ; Assmann, U.: DeltaEcore - A Model-Based Delta Language Generation Framework. In: *Proc. zur Modellierung*, GI, 2014 19, 121, 160

[SSJ14] Schatz, A. ; Schöllhammer, O. ; Jäger, J.: Ansatz zum Umgang mit Komplexität in Unternehmen. In: *Controlling* 26 (2014), Nr. 12, S. 686–693 4, 112, 113

[SSM16] Schneider, A. W. ; Schweda, C. M. ; Matthes, F.: Identifikation überalterter Komponenten in einer IT-Architektur. In: *Proc. der Multikonferenz Wirtschaftsinformatik (MKWI)*, Universitätsverlag Ilmenau, 2016, S. 1465–1476 65

[SSS15] Shatnawi, A. ; Seriai, A. ; Sahraoui, H.: Recovering Architectural Variability of a Family of Product Variants. In: *Proc. der Intl. Conference on Software Reuse (ICSR)*, Springer, 2015 (Lecture Notes in Computer Science), S. 17–33 3, 70

[Sti07] Stirling, A.: A General Framework for Analysing Diversity in Science, Technology and Society. In: *Interface* 4 (2007), Nr. 15, S. 707–719 66

[Str98] Strahringer, S.: Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips. In: *Proc. zur Modellierung*, GI, 1998, S. 1–6 37

[SW11] Solis, C. ; Wang, X.: A Study of the Characteristics of Behaviour Driven Development. In: *Proc. der Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2011, S. 383–387 76, 77

[SWG13] Schütz, A. ; Widjaja, T. ; Gregory, R. W.: Escape from Winchester Mansion – Toward a Set of Design Principles to Master Complexity in

IT Architectures. In: *Proc. der Intl. Conference on Information Systems (ICIS)*, AIS, 2013, S. 1–19 vii, 22, 23, 112

[SWK13] Schütz, A. ; Widjaja, T. ; Kaiser, J.: Complexity In Enterprise Architectures – Conceptualization and Introduction of a Measure From a System Theoretic Perspective. In: *Proc. der European Conference on Information Systems (ECIS)*, AIS, 2013, S. 1–12 4, 64

[SWS13] Schmidt, C. ; Widjaja, T. ; Schütz, A.: Messung der Komplexität von IT-Landschaften auf der Basis von Architektur-Metamodellen: Ein generischer Ansatz und dessen Anwendung im Rahmen der Architektur-Transformation. In: *INFORMATIK – Beiträge der Jahrestagung der Gesellschaft für Informatik*, GI, 2013 64

[SWS16] Seidl, C. ; Winkelmann, T. ; Schaefer, I.: A Software Product Line of Feature Modeling Notations and Cross-Tree Constraint Languages. In: *Proc. zur Modellierung*, GI, 2016 16

[SZM14] Schneider, A. W. ; Zec, M. ; Matthes, F.: Adpting Notions of Complexity for Enterprise Architecture Management. In: *Proc. der Americas Conference on Information Systems (AMCIS)*, AIS, 2014, S. 822–831 63

[The11] The Open Group: *TOGAF Version 9.1*. Van Haren Publishing, 2011 vii, 2, 22, 23, 24

[Tie16] Tiemeyer, E.: *Handbuch IT-Systemmanagement: Handlungsfelder, Prozesse, Managementinstrumente, Good-Practices*. München : Carl Hanser, 2016 2

[TW01] Taveter, K. ; Wagner, G.: Agent-Oriented Enterprise Modeling Based on Business Rules. In: *Proc. der Intl. Conference on Conceptual Modeling (ER)*, Springer, 2001, S. 527–540 75

[VBD+13] Voelter, M. ; Benz, S. ; Dietrich, C. ; Engelmann, B. ; Helander, M. ; Kats, L. ; Visser, E. ; Wachsmuth, G.: *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. CreateSpace Independent Publishing Platform, 2013 29

[vKV00] van Deursen, A. ; Klint, P. ; Visser, J.: Domain-Specific Languages – An Annotated Bibliography. In: *ACM SIGPLAN Notices* 35 (2000), Nr. 6, S. 26–36 29

[VST07] Vasconcelos, A. ; Sousa, P. ; Tribolet, J.: Information System Architecture Metrics: an Enterprise Engineering Evaluation Approach. In: *The Electronic Journal Information Systems Evaluation* 10 (2007), Nr. 1, S. 91–122 65

[vvMP14] van Zee, M. ; van der Linden, D. ; Marosin, D. ; Plataniotis, G.: Formalizing Enterprise Architecture Decision Models using Integrity Constraints. In: *Proc. der Conference on Business Informatics (CBI)*, IEEE, 2014, S. 143–150 153

[WF06] Winter, R. ; Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. In: *Proc. der Intl. Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, IEEE, 2006, S. 30–37 22, 24

[Wil18] Wille, D.: *Custom-Tailored Product Line Extraction, Dissertation.* Braunschweig : TU Braunschweig, 2018 20, 21, 43, 68, 69, 159

[WKTB12] Widjaja, T. ; Kaiser, J. ; Tepel, D. ; Buxmann, P.: Heterogeneity in IT Landscapes and Monopoly Power of Firms: A Model to Quantify Heterogeneity. In: *Proc. der Intl. Conference on Information Systems (ICIS)*, AIS, 2012, S. 1–14 64

[WRSS17] Wille, D. ; Runge, T. ; Seidl, C. ; Schulze, S.: Extractive Software Product Line Engineering Using Model-Based Delta Module Generation. In: *Proc. des Intl. Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, ACM, 2017, S. 36–43 20, 21, 43

[WS17] Wehling, K. ; Schaefer, I.: Towards an Expert System for Identifying and Reducing Unnecessary Complexity of IT Architectures. In: *INFORMATIK – Beiträge der Jahrestagung der Gesellschaft für Informatik*, GI, 2017, S. 1523–1529 73

[WSS16] Wille, D. ; Schulze, S. ; Schaefer, I.: Variability Mining of State Charts. In: *Proc. des Intl. Workshop on Feature Oriented Software Development (FOSD)*, ACM, 2016, S. 63–73 20, 69

[WSSS16] Wille, D. ; Schulze, S. ; Seidl, C. ; Schaefer, I.: Custom-Tailored Variability Mining for Block-Based Languages. In: *Proc. der Intl. Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2016, S. 271–282 20, 21, 43, 69, 70

[WWPS16] Wehling, K. ; Wille, D. ; Pluchator, M. ; Schaefer, I.: Towards Reducing the Complexity of Enterprise Architectures by Identifying Standard Variants Using Variability Mining. In: *Proc. des Automobil Symposium Wildau (ASW)*, THW, 2016, S. 37–43 33

[WWS+17] Wille, D. ; Wehling, K. ; Seidl, C. ; Pluchator, M. ; Schaefer, I.: Variability Mining of Technical Architectures. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, ACM, 2017, S. 39–48 33

[WWSS17a] Wehling, K. ; Wille, D. ; Seidl, C. ; Schaefer, I.: Automated Recommendations for Reducing Unnecessary Variability of Technology Architectures. In: *Proc. des Intl. Workshop on Feature Oriented Software Development (FOSD)*, ACM, 2017, S. 1–10 73

[WWSS17b] Wehling, K. ; Wille, D. ; Seidl, C. ; Schaefer, I.: Decision Support for Reducing Unnecessary IT Complexity of Application Architectures. In: *Proc. der Intl. Conference on Software Architecture Workshops (ICSAW)*, IEEE, 2017, S. 161–168 33

[WWSS18] Wehling, K. ; Wille, D. ; Seidl, C. ; Schaefer, I.: Reducing Variability of Technically Related Software Systems in Large-Scale IT Landscapes. In: *Proc. der Intl. Conference on Computer Science and Software Engineering (CASCON)*, ACM, 2018, S. 224–235 117

[Yue14] Yue, J.: Transition from EBNF to Xtext. In: *Proc. der Student Research Competition im Rahmen der Intl. Conference on Model Driven Engineering Languages and Systems (PSRC@MODELS)*, CEUR-WS, 2014, S. 75–80 v, 30, 31

[ZHMP11] Zhang, X. ; Haugen, Ø. ; Møller-Pedersen, B.: Model Comparison to Synthesize a Model-Driven Software Product Line. In: *Proc. der Intl. Systems and Software Product Line Conference (SPLC)*, IEEE, 2011, S. 90–99 3, 4, 69, 70

[ZHMP12] Zhang, X. ; Haugen, Ø. ; Møller-Pedersen, B.: Augmenting Product Lines. In: *Proc. der Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, 2012, S. 766–771 69

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Braunschweig, 26. April 2019

_____

A Anhang

A.1 Domänenspezifische Sprache zur Abbildung von Geschäftsregeln

In Abschnitt 4.1 wurde eine DSL vorgestellt, welche sich zur formalen Beschreibung von Geschäftsanforderungen und -zielen in Form von Geschäftsregeln eignet, um damit eine automatisierte Analyse von Kandidaten für unnötige Variabilität aus einer businessorientierten Perspektive zu ermöglichen. Nachfolgend wird die entwickelte DSL vollständig aufgeführt.

Das folgende Listing A.1 zeigt die grobe Struktur unserer DSL für Geschäftsregeln.

```
Rule:
   'RULE' id = STRING ('description' title = STRING)?
       givenRule = GivenRulePart
       whenRule = WhenRulePart
       thenRule = ThenRulePart
       (weightRule = WeightRulePart)?
;

[..]
```

Listing A.1: Syntax der groben Struktur einer Geschäftsregel

Das nachfolgende Listing A.2 zeigt die Syntax des `GivenRulePart` unserer DSL.

```
[..]

GivenRulePart:
   'GIVEN' informationElement += InformationElement*
      'AND' applicationConditions += ApplicationCondition
      ('OR' applicationConditions += ApplicationCondition)*
;

[..]
```

Listing A.2: Grammatikausschnitt für den `GivenRulePart`

Das Listing A.3 stellt die Syntax des Nicht-Terminals `InformationElement` aus dem `GivenRulePart` dar.

```
[..]

InformationElement:
   'information' 'on' neededInformation
      += NeededInformationElement
      (',' neededInformation += NeededInformationElement)*
;

NeededInformationElement:
   VariabilityInformationElement
      | UsageFrequencyInformationElement
      | FeatureInformationElement
      | DependencyInformationElement
      | StrategicFitInformationElement
      | TCOInformationElement
      | RestructuringCostInformationElement
;

VariabilityInformationElement:
   {VariabilityInformationElement} 'variability'
;

UsageFrequencyInformationElement:
   {UsageFrequencyInformationElement} 'frequencies'
;

[..]
```

Listing A.3: Grammatikausschnitt für das `InformationElement`

Das folgende Listing A.4 zeigt die Syntax des Nicht-Terminals `ApplicationCondition` aus dem `GivenRulePart` unserer DSL.

```
[..]

ApplicationCondition:
   SingleApplicationCondition
      | GroupApplicationCondition
;

SingleApplicationCondition:
   {SingleApplicationCondition} 'singleElement'
;

GroupApplicationCondition:
   {GroupApplicationCondition} 'variabilityGroup'
      groupVariability = GroupVariability
;

GroupVariability:
   AlternativeGroupVariability
      | RelatedGroupVariability
;

AlternativeGroupVariability:
   {AlternativeGroupVariability} 'alternative'
;

RelatedGroupVariability:
   {RelatedGroupVariability} 'related'
;

[..]
```

Listing A.4: Grammatikausschnitt für die `ApplicationCondition`

Das nachfolgende Listing A.5 zeigt die Grammatik für den `WhenRulePart` unserer DSL für Geschäftsregeln.

```
[..]

WhenRulePart:
        'WHEN' expression = Expression
;

Expression:
        OrExpression
;

OrExpression returns Expression:
        AndExpression ({OrExpression.left = current} 'OR' right
            = AndExpression)*
;

AndExpression returns Expression:
        WithExpression ({AndExpression.left = current} 'AND'
           right = WithExpression)*
;

WithExpression returns Expression:
        RelationalExpression ({WithExpression.left = current} '
           WITH' right = RelationalExpression)*
;

RelationalExpression returns Expression:
        HasCharacteristicExpression ({RelationalExpression.left
            = current} operator = RelationalOperator right =
           HasCharacteristicExpression)*
;

HasCharacteristicExpression returns Expression:
        UnaryExpression ({HasCharacteristicExpression.left =
           current} 'has' right = UnaryExpression)*
;

enum RelationalOperator:
        EQUAL = '=' | NOT_EQUAL = '!=' |
        SMALLER = '<' | SMALLER_OR_EQUAL = '<=' |
        GREATER = '>' | GREATER_OR_EQUAL = '>='
;

[..]
```

Listing A.5: Grammatikausschnitt für den `WhenRulePart`

Im Listing A.6 ist die Syntax für den ersten Teil des Nicht-Terminals `UnaryExpression` aus dem `WhenRulePart` unserer DSL dargestellt.

```
[..]

UnaryExpression returns Expression:
        SelectionElement | VariabilityElement |
           StrategicFitElement | StringElement |
           IntegerElement | ParenthesesExpression |
           CharacteristicElement
;

SelectionElement:
        ModelSelectionElement | DisparityRelationElement
;

VariabilityElement:
        variability = VariabilityType
;

enum VariabilityType:
        MANDATORY = 'mandatory' | ALTERNATIVE = 'alternative' |
            OPTIONAL = 'optional'
;

StrategicFitElement:
        GroupStandardFit | LimitedStandardFit |
           StandardizationCandidateFit |
           NotGroupwideDefinedFit | FreezeFit
;

GroupStandardFit:
        {GroupStandardFit} 'groupStandard'
;

LimitedStandardFit:
        {LimitedStandardFit} 'limitedStandard'
;

StandardizationCandidateFit:
        {StandardizationCandidateFit} 'standardizationCandidate
           '
;

NotGroupwideDefinedFit:
        {NotGroupwideDefinedFit} 'notGroupwideDefined'
;

[..]
```

Listing A.6: Teil 1 des Grammatikausschnitt für die `UnaryExpression`

Das Listing A.7 führt die Syntax des Nicht-Terminals `UnaryExpression` aus dem `WhenRulePart` fort.

```
[..]

FreezeFit:
        {FreezeFit} 'freeze'
;

StringElement:
        value = STRING
;

IntegerElement:
        ExtremeElement | IntElement
;

ExtremeElement:
        extreme = Extreme
;

enum Extreme:
        MINIMUM = 'minimum' | MAXIMUM = 'maximum'
;

IntElement:
        value = INT
;

ParenthesesExpression:
        '(' expression = Expression ')'
;

[..]
```

Listing A.7: Teil 2 des Grammatikausschnitts für die `UnaryExpression`

Das nachfolgende Listing A.8 stellt den ersten Teil der Grammatik für das Nicht-Terminal `CharacteristicElement` des `WhenRulePart` der vorgestellten DSL dar. Dabei ist das `CharacteristicElement` der letzte Teil der `UnaryExpression`.

```
[..]

CharacteristicElement:
        PredefinedCharacteristicElement |
           CustomCharacteristicElement
;

PredefinedCharacteristicElement:
        NameCharacteristicElement |
           VersionCharacteristicElement |
           SimilarityCharacteristicElement |
           FrequencyCharacteristicElement |
        DependencyCharacteristicElement |
           FeatureCharacteristicElement |
           StrategicFitCharacteristicElement |
           TCOCharacteristicElement |
        RestructuringCostCharacteristicElement |
           VariabilityCharacteristicElement |
           DistanceCharacteristicElement |
           NumberOfFeaturesCharacteristicElement
;

NameCharacteristicElement:
        {NameCharacteristicElement} 'name'
;

VersionCharacteristicElement:
        {VersionCharacteristicElement} 'version'
;

SimilarityCharacteristicElement:
        {SimilarityCharacteristicElement} 'similarity'
;

DistanceCharacteristicElement:
        {DistanceCharacteristicElement} 'distance'
;

VariabilityCharacteristicElement:
        {VariabilityCharacteristicElement} 'variability'
;

[..]
```

Listing A.8: Teil 1 der Grammatik für das `CharacteristicElement`

Im folgenden Listing A.9 wird der zweite Teil der Grammatik für das Nicht-Terminal `CharacteristicElement` des `WhenRulePart` unserer DSL fortgeführt.

```
[..]

FrequencyCharacteristicElement:
        ModelFrequencyCharacteristicElement |
           IndividualFrequencyCharacteristicElement
;

ModelFrequencyCharacteristicElement:
        {ModelFrequencyCharacteristicElement} 'modelFrequency'
;

IndividualFrequencyCharacteristicElement:
        {IndividualFrequencyCharacteristicElement} '
           individualFrequency'
;

DependencyCharacteristicElement:
        {DependencyCharacteristicElement} 'dependency'
;

FeatureCharacteristicElement:
        {FeatureCharacteristicElement} 'feature'
;

NumberOfFeaturesCharacteristicElement:
        {NumberOfFeaturesCharacteristicElement} '
           numberOfFeatures'
;

StrategicFitCharacteristicElement:
        {StrategicFitCharacteristicElement} 'strategicFit'
;

TCOCharacteristicElement:
        {TCOCharacteristicElement} 'TCO'
;

RestructuringCostCharacteristicElement:
        {RestructuringCostCharacteristicElement} '
           restructuringCosts'
;

CustomCharacteristicElement:
        'customCharacteristic' customCharacteristic = STRING
;

[..]
```

Listing A.9: Teil 2 der Grammatik für das `CharacteristicElement`

Das Listing A.10 zeigt die Grammatik für den `ThenRulePart` unserer DSL.

```
[..]

ThenRulePart:
        'THEN' decisionElement = DecisionElement
;

DecisionElement:
        element = [ModelSelectionElement] 'is' decision =
           Decision
;

ModelSelectionElement:
        logicalElement | physicalElement
;

enum Decision:
        REQUIRED = 'required' | NOTREQUIRED = 'notRequired'
;

logicalElement:
        'logicalCandidate' ('in' disparityRelationElement = [
           DisparityRelationElement])?
;

physicalElement:
        'physicalCandidate'
;

DisparityRelationElement:
        'disparity' 'relation' name = ID
;

[..]
```

Listing A.10: Grammatikausschnitt für den `ThenRulePart`

Das folgende Listing A.11 zeigt die Grammatik für den `WeightRulePart`, welcher der letzte Teil unserer DSL ist.

```
[..]

WeightRulePart:
        'WEIGHT' weightingElement = WeightingElement
;

WeightingElement:
        value = INT
;
```

Listing A.11: Grammatikausschnitt für den `WeightRulePart`

A.2 Deltadialekt mit domänenspezifischen Deltaoperationen

Im Unterabschnitt 5.1.1 wurde der Deltadialekt für domänenspezifische Deltaoperationen vorgestellt, mit deren Hilfe ein Ist-150%-Modell auf Basis einer spezifischen Restrukturierungsempfehlung automatisiert in das resultierende Soll-150%-Modell transformiert werden kann. In den folgenden Listings A.12 bis A.13 sind alle entwickelten Deltaoperationen für diesen Deltadialekt aufgeführt.

```
deltaDialect
{
   configuration:
      metaModel: <http://www.tu-braunschweig.de/isf/
         familymining/ea/ta>;

   deltaOperations:
      //EAModel
      removeOperation removePhysicalElementFromEAModel(
         PhysicalElement value, EAModel [physicalElements]
         element);
      removeOperation removeLogicalElementFromEAModel(
         LogicalElement value, EAModel [logicalElements]
         element);
      removeOperation removeCategoryFromEAModel(Category value,
          EAModel [categories] element);

      //Model
      removeOperation removeModelElementFromModel(ModelElement
         value, Model [modelElements] element);

      //ModelElement
      customOperation removeContainingModelFromModelElement(
         String value, ModelElement element);
      customOperation addContainingModelToModelElement(String
         value, ModelElement element);
      customOperation
         removeRelatedVariabilityGroupIdFromModelElement(
         String value, ModelElement element);
      customOperation
         addRelatedVariabilityGroupIdToModelElement(String
         value, ModelElement element);

      [..]
```

Listing A.12: Deltaoperationen des entwickelten Deltadialektes - Teil 1

Listing A.13 zeigt den zweiten Teil des entwickelten Deltadialektes.

```
        [..]

        customOperation
           modifyRelatedVariabilityGroupIdOfModelElement(String
           oldValue, String newValue, ModelElement element);
        customOperation removeRelatedGroupFlagFromModelElement(
           String relatedGroupFlag, ModelElement element);
        modifyOperation modifyVariabilityOfModelElement(
           Variability value, ModelElement [variability] element
           );
        modifyOperation modifyVariabilityGroupIdOfModelElement(
           String value, ModelElement [variabilityGroupId]
           element);

        //ModelDimension
        removeOperation removeModelElementFromModelDimension(
           ModelElement value, ModelDimension [
           containedModelElements] element);

        //PhysicalElement
        removeOperation removeModelElementFromPhysicalElement(
           ModelElement value, PhysicalElement [
           containingModelElements] element);

        //LogicalElement
        removeOperation removePhysicalElementFromLogicalElement(
           PhysicalElement value, LogicalElement [
           physicalElements] element);

        //Category
        removeOperation removePhysicalElementFromCategory(
           PhysicalElement value, Category [
           containedPhysicalElements] element);
}
```

Listing A.13: Deltaoperationen des entwickelten Deltadialektes - Teil 2

A.3 Details der Implementierung

Im Abschnitt 6.1 wurde die Architektur unseres implementierten Softwareprototypens auf der Ebene von Oberkomponenten gezeigt (vgl. Abbildung 6.1). In diesem Abschnitt erfolgt die Beschreibung der Implementierung unseres Prototypens noch detaillierter. Hierzu zeigen wir zuerst ein *Komponentendiagramm* in Abbildung A.1, welches alle entwickelten Einzelkomponenten, also entsprechende Java-Plugins, unseres Softwareprototypen enthält. Die folgende Tabelle A.1 gibt dabei Aufschluss über die Zuordnung dieser Komponenten zu den sechs Oberkomponenten aus Abbildung 6.1. Diese Oberkomponenten sind auch in Abbildung A.1 dargestellt und können anhand ihrer Buchstaben (A–F) aus der Tabelle A.1 identifiziert werden.

Unterkomponente (Plugin)	**Oberkomponente**
Data Cleaning	Variability Analysis (A)
Variability Determination	Variability Analysis (A)
Business Analysis	Recommendation Deriviation (B)
Technology Analysis	Recommendation Deriviation (B)
Recommendation Determination	Recommendation Deriviation (B)
Single Assessment	Recommendation Assessment (C)
Portfolio Assessment	Recommendation Assessment (C)
Business Rule DSL	DSL Provider (D)
Graph Description DSL	DSL Provider (D)
Dependency DSL	DSL Provider (D)
Effort DSL	DSL Provider (D)
Delta Language	DSL Provider (D)
Simulation	Simulation & Restructuring (E)
Restructuring	Simulation & Restructuring (E)
Views	Viewer (F)

Tabelle A.1: Zuordnung von Komponenten zu Oberkomponenten

Im folgenden wird für jedes in Abbildung A.1 abgebildete Java-Plugin eine kurze Beschreibung seiner Funktion, der zusätzlich eingesetzten Technologien sowie der benötigten Schnittstellen gegeben. Ist dabei für solch eine Komponente keine Technologie angegeben, so wurde ihre Funktionalität ohne zusätzliche Bibliotheken oder Plugins mit reinen Java-Mitteln implementiert.

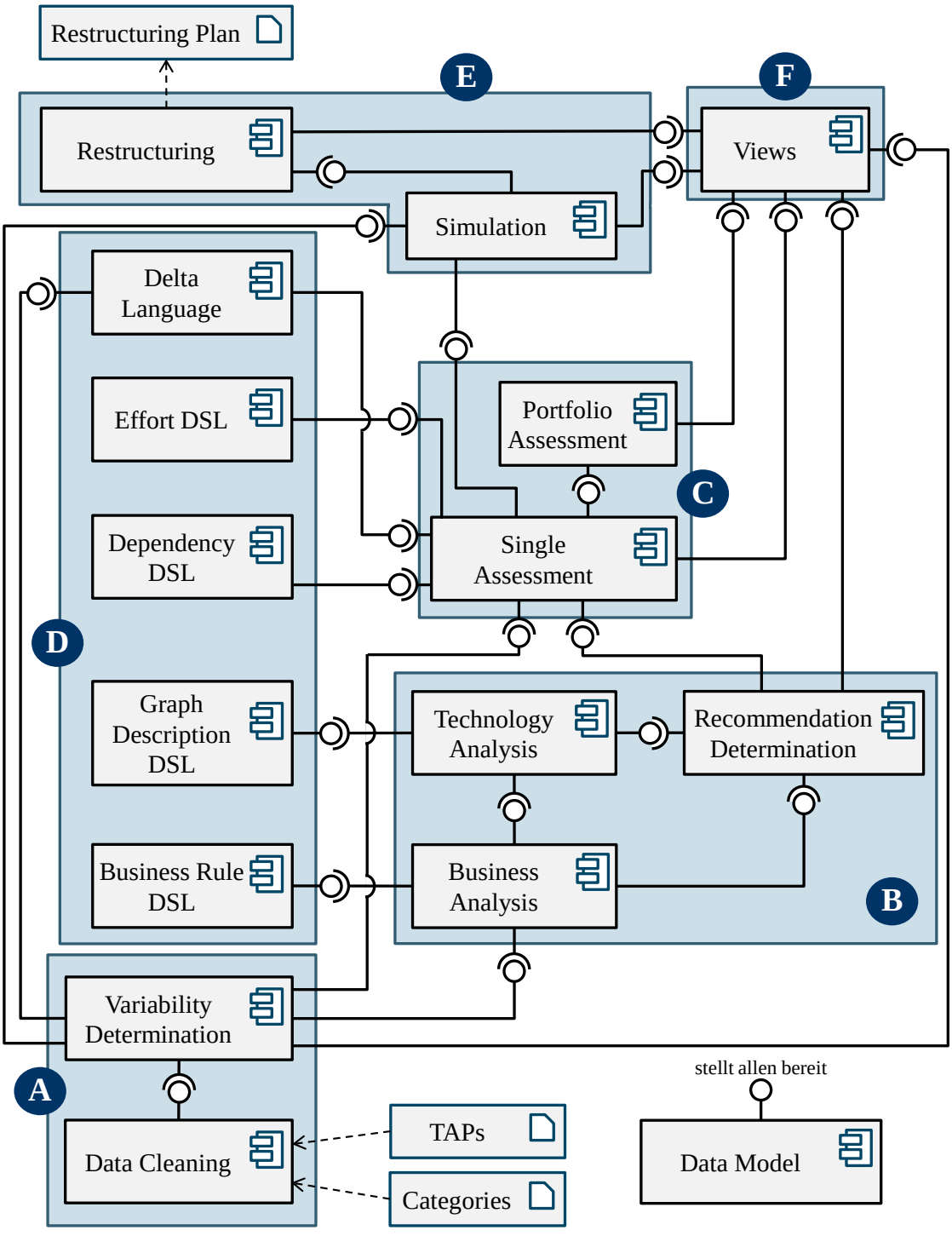


Abbildung A.1: Komponentendiagramm der prototypischen Implementierung

Data Cleaning	
Funktion:	Stellt geeignete CSV-Importer für die Datentransformation (vgl. Unterabschnitt 3.1.2) von `TAPs` und den dazugehörigen `Categories` bereit. Realisiert die Datenaufbereitung für die ausgewählten TAPs mittels Strukturanalyse und Ausreißersuche (vgl. Unterabschnitt 3.1.3).
Technologie:	*EMF* zur Modellierung von Ecore-Metamodellen für die Verarbeitung von strukturierten Datenmodellen
Benötigte Schnittstellen:	keine, benötigt aber die beiden zu importierenden CSV-Dateien `TAPs` und `Categories`

Variability Determination	
Funktion:	Realisierung der Variabilitätsbestimmung (vgl. Abschnitt 3.2) für aufbereitete TAPs
Technologie:	–
Benötigte Schnittstellen:	1. *Data Cleaning*: Stellt die aufbereiteten TAPs in Form von strukturierten Datenmodellen sowie deren Kategorien zur Verfügung

Business Analysis	
Funktion:	Identifizierung von Kandidaten für unnötige Variabilität (vgl. Abschnitt 3.3) und Umsetzung der regelbasierten Businessanalyse (vgl. Abschnitt 4.1) für diese Kandidaten zur Ermittlung von Restrukturierungspotentialen
Technologie:	–
Benötigte Schnittstellen:	1. *Variability Analysis*: Stellt das generierte 150%-Modell für die analysierten TAPs bereit 2. *Business Rule DSL*: Liefert die spezifizierten Geschäftsregeln für die regelbasierte Analyse

Business Rule DSL	
Funktion:	DSL zur Spezifizierung von Geschäftsregeln für die regelbasierte Businessanalyse (vgl. Unterabschnitt 4.1.1)
Technologie:	*Xtext* zur Entwicklung von Domänenspezifischen Sprachen
Benötigte Schnittstellen:	keine

Technology Analysis	
Funktion:	Realisierung der graphbasierten Technologieanalyse (vgl. Abschnitt 4.2) zum Vergleich von VKGs und der Identifizierung von technischen Abhängigkeiten
Technologie:	–
Benötigte Schnittstellen:	1. *Business Analysis*: Liefert die identifizierten Restrukturierungspotentiale 2. *Graph Description DSL*: Stellt die spezifizierte Graphbeschreibung für die technische Analyse bereit

Graph Description DSL	
Funktion:	DSL zur Spezifizierung von Graphbeschreibungen für die graphbasierte Technologieanalyse (vgl. Unterabschnitt 4.2.1)
Technologie:	*Xtext* zur Entwicklung von Domänenspezifischen Sprachen
Benötigte Schnittstellen:	keine

Recommendation Determination	
Funktion:	Implementierung der Ableitung konkreter Restrukturierungsempfehlungen (vgl. Abschnitt 4.3)
Technologie:	–
Benötigte Schnittstellen:	1. *Business Analysis*: Liefert die identifizierten Restrukturierungspotentiale 2. *Technology Analysis*: Stellt die VKGs, deren Ähnlichkeiten und die technischen Abhängigkeiten für identifizierte Restrukturierungspotentiale bereit

Effort DSL	
Funktion:	DSL zur Spezifizierung von Aufwandsmatrizen (vgl. Unterabschnitt 5.1.3)
Technologie:	*Xtext* zur Entwicklung von Domänenspezifischen Sprachen
Benötigte Schnittstellen:	keine

Delta Language	
Funktion:	Delta Sprache zur Spezifizierung von anwendbaren Transformationsoperationen für das 150%-Modell (vgl. Unterabschnitt 5.1.1)
Technologie:	*Delta Ecore* zur Spezifizierung der erforderlichen Deltaoperationen
Benötigte Schnittstellen:	1. *Variability Analysis*: Stellt das Metamodell eines 150%-Modells zur Verfügung, um hierfür spezifische Deltaoperationen bestimmen zu können

Single Assessment	
Funktion:	Realisierung der Einzelbewertung von Restrukturierungsempfehlungen (vgl. Abschnitt 5.1)
Technologie:	*Delta Ecore* zur Generierung von spezifischen Deltaoperationen für die einzelnen Restrukturierungsempfehlungen
Benötigte Schnittstellen:	1. *Delta Language*: Stellt die Deltasprache für die Generierung von Deltaoperationen für die Differenzanalyse bereit 2. *Effort DSL*: Liefert die spezifizierte Aufwandsmatrix für die Aufwandsschätzung 3. *Dependency DSL*: Liefert das spezifizierte Abhängigkeitsmodell für die Anpassungsbedarfsanalyse 4. *Variability Analysis*: Stellt das generierte 150%-Modell für die Differenzanalyse zur Verfügung 5. *Recommendation Deriviation*: Bietet das Set aller abgeleiteten Restrukturierungsempfehlungen an

Portfolio Assessment	
Funktion:	Umsetzung der Portfoliobewertung für das Set aller einzeln bewerteten Restrukturierungsempfehlungen (vgl. Abschnitt 5.2)
Technologie:	–
Benötigte Schnittstellen:	1. *Single Assessment*: Liefert das Set aller einzeln bewerteten Restrukturierungsempfehlungen

Simulation	
Funktion:	Implementierung der Simulation von ausgewählten Restrukturierungsempfehlungen (vgl. Unterabschnitt 5.3.2)
Technologie:	*Delta Ecore* zur automatisierten Transformation des 150%-Modells
Benötigte Schnittstellen:	1. *Variability Analysis*: Stellt das generierte 150%-Modell für die Transformation zur Verfügung 2. *Single Assessment*: Liefert das Set aller einzeln bewerteten Restrukturierungsempfehlungen inklusive ihrer erforderlichen Deltaoperationen

Dependency DSL	
Funktion:	DSL zur Spezifizierung von Abhängigkeitsmodellen (vgl. Unterabschnitt 5.1.2)
Technologie:	*Xtext* zur Entwicklung von Domänenspezifischen Sprachen
Benötigte Schnittstellen:	keine

Views	
Funktion:	Grafische Oberflächen zur Darstellung der erzeugten Informationen und zur Interaktion mit dem Nutzer
Technologie:	*GEF* zur Erstellung von grafischen Oberflächen für Java-Applikationen, *JUNG* zur Visualisierung von Graphen
Benötigte Schnittstellen:	1. *Variability Analysis*: Stellt das generierte 150%-Modell der Ist-Architektur bereit 2. *Business Analysis*: Liefert die Ergebnisse der regelbasierten Businessanalyse 3. *Technology Analysis*: Bietet die Ergebnisse der graphbasierten Technologieanalyse an 4. *Single Assessment*: Stellt die Ergebnisse der Einzelbewertung zur Verfügung 5. *Portfolio Assessment*: Stellt die Ergebnisse der Portfoliobewertung zur Verfügung 6. *Simulation*: Liefert das transformierte 150%-Modell der Soll-Architektur 7. *Restructuring*: Bietet den erzeugten Restrukturierungsplan an

Restructuring	
Funktion:	Realisierung der Restrukturierungsplanung (vgl. Abschnitt 5.4) zur Erzeugung eines konkreten Restrukturierungsplans im HTML-Ausgabeformat
Technologie:	*HTML* zur Erstellung eines strukturierten Dokumentes für die spätere Darstellung
Benötigte Schnittstellen:	1. *Simulation*: Stellt das transformierte 150%-Modell und die umgesetzten Restrukturierungsempfehlungen bereit

Data Model	
Funktion:	Stellt das Metamodell für TAPs und generische Datenstrukturen (z.B. für die Restrukturierungsempfehlung) bereit, welche von allen anderen Komponenten genutzt werden können
Technologie:	*EMF* zur Modellierung von Ecore-Metamodellen für die Verarbeitung von strukturierten Datenmodellen
Benötigte Schnittstellen:	keine

A.4 Regelkatalog für die Fallstudie B

Für die Durchführung der Fallstudie B (vgl. Unterabschnitt 6.2.3) haben wir, gemeinsam mit Experten unseres Industriepartners, zehn verschiedene Geschäftsregeln mit Hilfe der entworfenen DSL (vgl. Unterabschnitt 4.1.1) spezifiziert und in einem Regelkatalog zusammengefasst. Die folgende Tabelle A.2 zeigt diesen Katalog, welcher Geschäftsregeln mit unterschiedlichen Hauptkriterien (z.B. *Häufigkeit*, *Variabilität* und *Funktionen*) für verschiedene Kandidatentypen (*Relationsgruppe (RG)*, *Alternativgruppe (AG)* und *unabhängige optionale Konfigurationsoption (uKO)*) enthält.

ID	Hauptkriterium	Kandidatentyp	Beschreibung
GR1	Häufigkeit	RG, AG	Standardize candidates in variability groups with usage frequency greater than 33%
GR2	Häufigkeit	RG, AG	Standardize candidates in variability groups with usage frequency greater than 66%
GR3	Häufigkeit	RG, AG	Standardize candidates in variability groups with maximum usage frequency
GR4	Häufigkeit	uKO	Remove independent optional candidates with usage frequency less than 10%
GR5	Variabilität	RG	Standardize candidates in related groups with mandatory variability
GR6	Funktionen	RG, AG	Remove candidates in variability groups where an element with similar features exists
GR7	Funktionen	RG, AG	Standardize candidates in variability groups with most features
GR8	Strategischer Fit	RG, AG	Standardize candidates in variability groups with strategic prioritization
GR9	TCO	RG, AG	Standardize candidates in variability groups with lowest Total Costs of Ownership (TCO)
GR10	Restrukturierungskosten	RG, AG	Standardize candidates in variability groups with lowest restructuring costs

RG=Relationsgruppe, AG=Alternativgruppe, uKO= unabhängige optionale Konfigurationsoption

Tabelle A.2: Regelkatalog

A.5 Fragen für die Experteninterviews

1. **Variabilitätsanalyse:**
   a) *Wie bewerten Sie die Verständlichkeit des vorgeschlagenen Ansatzes zur Variabilitätsanalyse?*
   b) *Wie bewerten Sie die Nützlichkeit des resultierenden 150%-Modells?*
2. **Ableitung von Restrukturierungsempfehlungen:**
   a) *Wie bewerten Sie die Verständlichkeit des vorgeschlagenen Ansatzes zur Businessanalyse?*
   b) *Wie bewerten Sie die Verständlichkeit des vorgeschlagenen Ansatzes zur Technologieanalyse?*
   c) *Wie bewerten Sie die Verständlichkeit der DSLs zur Spezifizierung von Geschäftsregeln und Graphbeschreibungen?*
   d) *Wie bewerten Sie die Nützlichkeit der resultierenden Restrukturierungsempfehlungen?*
3. **Bewertung und Restrukturierungsentscheidung:**
   a) *Wie bewerten Sie die Verständlichkeit des vorgeschlagenen Ansatzes zur Differenzanalyse?*
   b) *Wie bewerten Sie die Nützlichkeit von ermittelten Differenzwerten?*
   c) *Wie bewerten Sie die Verständlichkeit des vorgeschlagenen Ansatzes zur Abhängigkeitsanalyse?*
   d) *Wie bewerten Sie die Nützlichkeit von identifizierten Anpassungsabhängigkeiten?*
   e) *Wie bewerten Sie die Verständlichkeit des vorgeschlagenen Ansatzes zur Aufwandsschätzung?*
   f) *Wie bewerten Sie die Nützlichkeit von geschätzten Migrations- und Anpassungsaufwänden?*
   g) *Wie bewerten Sie die Nützlichkeit der Portfoliobewertung und Simulation von Restrukturierungsempfehlungen für die Entscheidungsfindung?*